10Tec iGrid for .NET 4.0
What's New in the Control

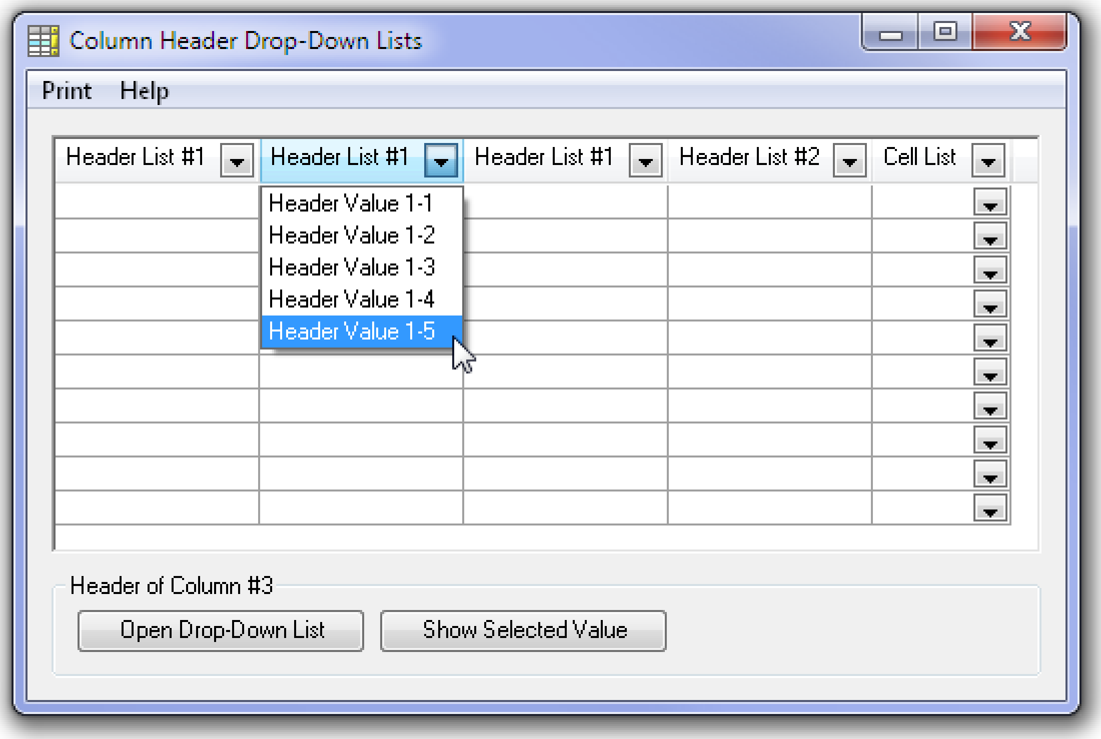
Contents

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;
- [Fixed] – a fixed bug or solved problem.

Drop-Down Controls for Column Headers

1. [New] This release of iGrid introduces the ability to attach drop-down controls to its column headers. This looks and works exactly the same like for the iGrid cells: if a cell has a drop-down control associated with it, a combo button is drawn inside the cell, and it can be used to open the drop-down control:



2. [New] You use the same technique to link a drop-down control to a column header like for the normal cells. If you assign an **iGDropDownList** object to the new **DropDownControl** property of a column header (**iGColHdr**) or its style (the **iGCol.ColHdrStyle.DropDownControl** property), the combo button is drawn in the column header and you can use it to open the attached drop-down list.

As a part of this redesign, the **iGStyleBase** class now implements the **DropDownControl** property, and both its descendants, **iGCellStyle** and **iGColHdrStyle**, simply inherits this property. No changes to your code are required because of this.

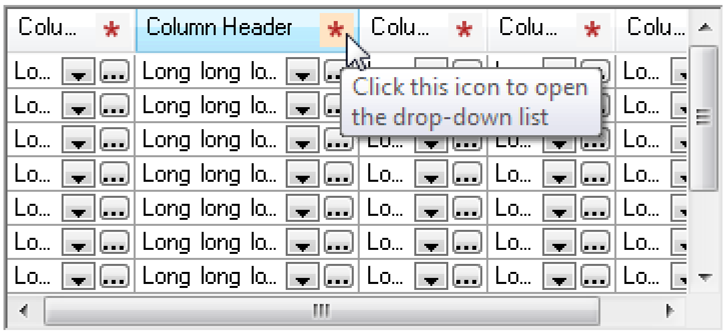
3. [New] Another part of the new functionality is the **AuxValue** property added to the column header object. It works exactly like the corresponding property of the cell object. It has the Object data type and is used to store the selected drop-down item. Sure, if you do not use drop-down lists in your column headers, you can use this property for your own purposes like a well-known **Tag** property in many .NET controls.

4. [New] To have the ability to detect whether the user is clicking the combo button in a column header, the new parameter **ElemControl** of the **iGElemControl** type was added to the **ColHdrMouseDown** event. It is set to **iGControl.ComboButton** if the mouse button is pressed over the combo button in a column header. You can use this value if you want to prevent the drop-down control to be opened in a particular column header – to do that, just set the **DoDefault** property of the event data to False.

5. [New] The **RequestEditColHdr**, **CancelEditColHdr** and **CommitEditColHdr** methods were implemented to have the ability to display column header drop-down controls programmatically like you can do that for the normal iGrid cells using the **RequedEditCurCell**, **CancelEditCurCell** and **CommitEditCurCell** methods.

6. [New] Three new events, **ColHdrComboBeforeCommit**, **ColHdrComboAfterCommit**, and **ColHdrComboCancel,** were implemented have the ability to respond to the user actions in column header drop-down lists. **ColHdrComboBeforeCommit** is raised when the user selected an item and iGrid is about to store the selected item in the **AuxValue** property of the corresponding column header. The **ColHdrComboAfterCommit** event is used when this value has been saved and the list has been closed. **ColHdrComboCancel** is raised when the user cancelled the selection in the drop-down list. In fact, these events work similar to the **BeforeCommitEdit**, **AfterCommitEdit**, and **CancelEdit** for the normal iGrid cells.

7. [New] iGrid allows to set custom tool tips for the column header combo buttons using the new **RequestColHdrComboButtonToolTipText** event:



By default, the combo button in a column header has the same tool tip as the whole column header, but this event is raised when the mouse pointer is placed into the area over the combo button and you can optionally set another special tool tip text for the combo button.

8. [New] To implement custom-drawn combo buttons in iGrid's column headers, use the new **CustomDrawColHdrComboButton** event.

The **DrawComboButton** method in the **IiGControlPaint** interface, used to draw cell combo buttons, is now also used to draw column header combo buttons. But it has the new Boolean

isHeader parameter you can use to distinguish whether this method is raised for the header or normal cells.

9. [New][Change] The **DrawColHdrContents** method has the new Boolean **drawComboButton** parameter you can use to specify whether the column header combo button should be drawn.

The new column header combo buttons functionality also touches other parts of iGrid, and this is reflected below. See the description for the **PrintCellControls** property you can use to control whether the column header combo buttons are printed, and the **RequestColHdrElemControlToolTipText** event you can use to set personal tool tips for column header combo buttons.

Other New Features and Changes in iGrid

1. [New][Change][Improved] In earlier versions, you should create some columns in the grid before you call the **GetPreferredRowHeight** method to calculate the optimal row height as the properties of the defined columns were used for that. In iGrid 4.0 you may not do that. Now **GetPreferredRowHeight** calculates the optimal row height using the properties of the default column (the **DefaultCol** object property) if there are no real columns in the grid.

   This feature is very useful in many situations. For instance, now you can set the optimal row height in the **DefaultRow.Height** property before you call the **FillWithData** method which automatically creates the corresponding column set in the grid. In earlier versions, you needed to loop through all rows and set their heights after you called **FillWithData** as **GetPreferredRowHeight** could return the required result only after the columns were created.

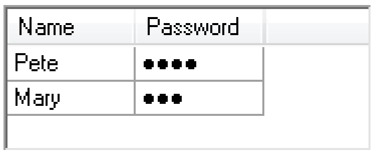
2. [New][Change][Improved] This release of iGrid implements a smart system that automatically sets the default row height to the optimal value which is best for displaying one line of text in cells. If you just drop the component from the toolbox on a form, the **DefaultRow.Height**, **DefaultRow.NormalCellHeight** and **DefaultAutoGroupRow.Height** properties are ignored and rewritten with the optimal values on the fly in the **ISupportIntialize.EndInit()** call after all properties of iGrid were initialized. This allows you not to worry about calculating the best row height in most cases when you work with textual information in the grid.

   iGrid internally calls the **GetPreferredRowHeight** method when calculating the best row height. This means that all important parameters which affect the text output are taken into account, including the OS’s current DPI setting, the grid font, the default column settings, the thickness of the horizontal grid line, etc.

   If you wish to specify a default row height explicitly using the **DefaultRow.Height, DefaultRow.NormalCellHeight** or **DefaultAutoGroupRow.Height** property, you can turn this smart mode off using the new **DefaultRowHeightAutoSet** Boolean property in the “Default Settings” property group. Its default value is True which defines the behavior described above.

   Note that this smart system works only for iGrid’s used in the Windows Forms designer as the optimal row height calculation is performed only in the **ISupportIntialize.EndInit()** call added by the Windows Forms designer automatically to the form initialization code. If you create iGrid from your code manually, you should use the **GetPreferredRowHeight** method to implement the same task.

3. [New] This version of iGrid implements password masking which can be enabled for individual cells. To do that, just specify the new **iGCellTypeFlags.Password** flag in the **TypeFlags** property for the required cells. For instance, the password column in the grid below



Name	Password
Pete	●●●●
Mary	●●●

was created with the following line of code:

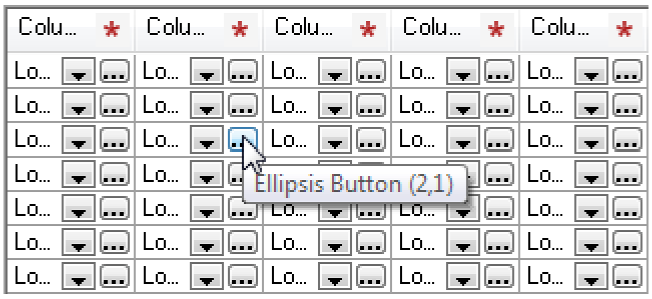
```
iGrid1.Cols.Add("Pwd", "Password").CellStyle.TypeFlags =
iGCellTypeFlags.Password;
```

By default, the system password character is used for masking, but you can redefine it with iGrid's new **PasswordChar** property of the Char data type. By default it contains the zero-code character which means that the system password character will be used.

4. [Renaming] The general member naming system in iGrid was corrected to become more consistent. All terms related to the whole iGrid control contains the word "Control" only (for instance, the **CustomControlPaint** property and the **IiGControlPaint** interface), but other terms related to small constituent cell and column header controls like check box, combo button are named using the "ElemControl" term which stands for "Elementary Control".

   As a result of this, the **iGControl** enumeration was renamed to **iGElemControl** and the **OverControl** parameter of a series of cell and mouse events was renamed to **ElemControl**. To upgrade the existing code, you just need to rename these two members if you use them.

5. [New] This version of iGrid introduces the ability to display personal tooltips for elementary controls in cells and column headers (combo buttons, ellipsis buttons, etc):



This is done with the help of the two new events, **RequestCellElemControlToolTipText** and **RequestColHdrElemControlToolTipText**, which have the same signature defined by the new **iGRequestElemControlToolTipTextEventHandler** delegate. It provides each event with the following data defined in an instance of the **iGRequestElemControlToolTipTextEventArgs** class:

```
public class iGRequestElemControlToolTipTextEventArgs : EventArgs
{
  public readonly int RowIndex;
  public readonly int ColIndex;
  public readonly iGElemControl ElemControl;
  public string Text;
}
```

The **RowIndex**/**ColIndex** properties indicate the cell or column header one of these events is raised for. The **ElemControl** indicates what cell elementary control is under the mouse pointer. The **Text** property passed to these events contains the text which is displayed in the tool tip for the whole cell or column header. If you need another special tool tip text for a cell elementary control, assign the desired string to the **Text** property in these events.

6. [New] The **RowVisibleChanged** event was implemented. It is raised when the **Visible** property of a row has been changed. The index of this row is passed in the **e.RowIndex** argument, and the Boolean **e.Visible** argument contains the new visibility state of the row. Note that this event is raised only when the **Visible** property has been really changed (for instance, if the same value is assigned to it, the event isn't raised). The event is also raised when the visibility of a row has been changed due to some internal operations, such as filtering the grid during incremental search.

7. [New][Change][Improved] In previous versions, you accessed the cell style properties defined for a particular row through a set of properties in the **iGRow** class – such as **CellType**, **BackColor**, **CellSelectable**, **Font**, etc. There were no strong naming system used for those properties because of step-by-step development of iGrid in different versions, so it was hard to say whether a property of the **iGRow** class is related to its cells or defines the look or behavior of the whole row (for instance, like the **Type** property).

   In this version all cell properties nested directly into the **iGRow** class were removed, and now they should be accessed only through the **CellStyle** object property. Thus previous calls like

   ```
   iGrid1.Rows[5].TextAlign = iGContentAlignment.BottomRight;
   ```

   should be transformed into the following one:

   ```
   iGrid1.Rows[5].CellStyle.TextAlign = iGContentAlignment.BottomRight;
   ```

   To minimize memory consumption and time of rows creation, these **CellStyle** objects (instances of the **iGCellStyle** class) are null references by default, but they are created on the fly when you access them. This is done for the convenience of coding - to avoid manual creation of **iGCellStyle** objects in code each time when you change a cell property in a row. This works similar to the iGrid cells, when the cell style object associated with a cell is created automatically when you change a property of the cell.

   There is only one drawback in this approach: in this scenario you cannot test whether a row's **CellStyle** object is a null reference (i.e. it was not assigned). To allow this, iGrid has the new Boolean **CreateRowCellStyleDynamically** property which determines whether a row's **CellStyle** is created automatically if needed. By default it's True, and this corresponds to the behavior described above. To prohibit automatic creation of the **CellStyle** objects in the **iGRow** instances, set this property to False.

8. [Improved] The events of iGrid were better categorized. New event groups were introduced for that. For instance, now all events related to the search-as-type functionality or tooltip dynamic request are gathered in separate event groups like "Search-as-Type" or "ToolTips". A series of former event groups (for instance, related to the header) were separated to smaller groups to make it easier to find necessary events ("Column Header Action", "Column Header Appearance", "Column Header Mouse", "Columns", etc).

9. [Improved] In previous versions the **FillWithData** method of iGrid has 4 overloaded versions for each possible data source (**DataTable**, **DataView**, **IDataReader**, **IDbCommand**) – 8 overloaded versions in all taking into account the optional **useCurColSet** parameter. To provide new data population features, 5 new optional parameters were added in this version (see the detailed description below). To avoid the huge number of overloaded versions of the method, the first strongly typed argument was converted to the general **Object** type to replace 4 overloaded versions with one. As the result, the existing 8 overloaded versions of the method are now represented with the following two ones:

```
FillWithData(object dataSource)
FillWithData(object dataSource, bool useCurColSet)
```

The same concerns the **iGDropDownList.FillWithData** method – the previous 8 versions were replaced with these 2:

```
FillWithData(object dataSource, string itemValueCol)
FillWithData(object dataSource, string itemValueCol, string itemTextCol)
```

No changes are required in the old code to keep the existing functionality.

10. [New] The **FillWithData** method of iGrid now allows you to populate row keys and row levels using the specified columns from the data source while adding the main data at one go. This is done with the following new overloaded version of the method:

    ```
    FillWithData(object dataSource, bool useCurColSet, string rowKeyCol,
    string rowLevelCol)
    ```

    If you need to use one of the columns as row keys (the **iGRow.Key** property), specify its name in the **rowKeyCol** parameter. Row level data (**iGRow.Level**) can be retrieved while populating the grid if you specify the corresponding column name in the **rowLevelCol** parameter. If you wish to use just one of these features, specify a null reference as the value of the unused parameter.

    If the **rowKeyCol** parameter is set and the type of the specified column isn't string, the values are converted to strings on the fly. If the grid's **UniqueKeys** property is set to True and a non-unique key is detected during population, iGrid raises the **ArgumentException** with the text "Key already exists".

    If the **rowLevelCol** parameter is specified, iGrid also automatically adds tree buttons to the hierarchical set of rows so they can be interactively expanded/collapsed after population. The row level column should have a numeric type which can be converted to System.Int32 – otherwise a type conversion exception will be thrown. To properly display the hierarchical structure, this column should contain row level values 0, 1, 2, etc set for the rows properly; in the other case the tree buttons in the rows can be displayed improperly.

    The columns specified in these 2 parameters aren't added to the grid if the grid column set is created automatically based on the specified data source (the **useCurColSet** parameter is False). If you need to display one of these columns or both in the grid, use this overloaded version of the method:

    ```
    FillWithData(object dataSource, bool useCurColSet, string rowKeyCol, bool
    addRowKeyCol, string rowLevelCol, bool addRowLevelCol, bool
    addTreeButtons)
    ```

    The Boolean **addRowKeyCol** and **addRowLevelCol** parameters allow to add the corresponding columns to the grid if required (specify True in the required parameter). The last Boolean **addTreeButtons** parameter is used to specify whether iGrid should add tree buttons automatically if the row level column is used while populating the grid.

11. [Renaming] The current release allows you to have combo buttons in column headers, and the new **CustomDrawColHdrComboButton** event is used for custom drawing in these buttons. In previous versions you used the **CustomDrawComboButton** event for the same purpose for cells, but to conform to the new ideology this event was renamed to **CustomDrawCellComboButton**.

    This also concerns the **CustomDrawEllipsisButtonBackground** and **CustomDrawEllipsisButtonForeground** events renamed to

CustomDrawCellEllipsisButtonBackground and **CustomDrawCellEllipsisButtonForeground** respectively.

The **EllipsisButtonClick** event was also renamed to **CellEllipsisButtonClick**.

12. [Renaming] The items of the **iGSortType** enumeration were reordered to have a more logical order when similar items are grouped next to each other. The **iGSortType.ByCustomer** item was renamed to a more proper name **iGSortType.Custom**.

13. [New] The **iGCustomSortEventArgs** class used as the argument of the **CustomSort** event triggered for the columns with sort type set to **iGSortType.Custom** was supplemented with two new fields, **RowIndex1** and **RowIndex2**. They return the row indices for the values passed in the **Value1** and **Value2** fields respectively.

14. [New] If you implemented the **IiGControlPaint** interface in your class to do custom drawing of some or all iGrid parts, you needed to implement the **ControlsForeColor**, **ControlsBackColor** and **ControlsDisabledForeColor** members in any case. Now you should do that only if you return the new **iGControlPaintFunctions.ControlsColors** flag as the result of the **IiGControlPaint.SupportedFunctions** method which makes it easier to implement custom styling.

15. [Improved] Previous versions of iGrid could create up to 6 hidden top-level windows per grid and every drop-down list to listen to such system changes as system color or window parameter changes via processing the corresponding WinAPI messages WM_SETTINGCHANGE, WM_THEMECHANGE, etc. This version was redesigned to avoid the creation of all those hidden windows. As a result, the current version of iGrid uses less system resources.

PrintManager Enhancements

1. [New] The **PrintCellControls** property of the iGrid PrintManager component now also affects how the iGrid header is printed. If it does not contain the **iGPrintCellControls.ComboButton** flag, the combo buttons in the header aren't printed.

2. [New] PrintManager supports the printing of cells set to display passwords with the new **iGCellTypeFlags.Password** flag.

Fixed Bugs

1. [Fixed] When a cell drop-down list was opened, the two-arrow resize cursor was displayed for the column header dividers while the resize operation was impossible (like any other operation outside of the drop-down list while it stays opened). The same effect was present for row headers and the special column header over the row headers area – they were highlighted and the two-arrow resize cursor was displayed for the row header dividers while a drop-down list was opened.

2. [Fixed] iGrid's **HotTracking** property didn't affect row headers (they were highlighted when the mouse hovered over them even if **HotTracking** was set to False).

3. [Fixed] The row header did not change its look after the user had switched the OS visual theme.

4. [Fixed] When you called the **GetPreferredRowHeight** method, iGrid raised the events **CellDynamicFormatting**, **CellDynamicStringFormat**, **RequestDropDownControl**, **RequestAutoCompleteControl** with the **rowIndex** property of the event data set to -1. This

could cause an Argument Exception or Out of Range error if you used this value to access the grid cells in the handlers of these events.

5. [Fixed] The auto-height operation for rows did not work properly in some cases – 1 pixel at the bottom of the cell texts could be cut off. The same concerns the **GetPreferredRowHeight** method.

6. [Fixed] The Apply button in the interactive Column and Row collection editors used at design-time was positioned a little bit incorrectly.