10Tec iGrid.NET 11.2
What's New in Control

Contents

Most Important Improvements Briefly

- New iGrid methods for adding rows with cell values can speed up code performance by 2 times.
- Now you can specify 5 different conditions when iGrid commits cell edits.
- The **BeforeCommitEdit** and **AfterCommitEdit** events provide the old and new cell data.
- Merged cell texts are now printed as true text so that they can be selected and copied from PDFs.

v11.2.0 | 2024-Mar-12 (Release)

1. [New][Change] After the user puts a cell into edit mode, they have several options to finish editing, which iGrid will interpret as an attempt to save possible changes to the cell. The most used of them are pressing Enter or clicking outside the cell. However, committing changes and raising the associated **BeforeCommitEdit**/**AfterCommitEdit** events are not always necessary in such cases. For example, if the user put a cell into edit mode just to copy some text from it, clicking outside the cell should not always commit edit that did not actually happen.

   This version of iGrid implements the new **CommitEditCondition** property allowing you to set the condition under which iGrid commits cell edit in such cases. This property accepts values from the following new enumeration:

```
enum iGCommitEditCondition
{
    Always,
    TextEdited,
    TextChanged,
    ValueChanged,
    Never
}
```

   The table below explains what every value means:

Always	iGrid always commits edit, even if nothing has changed in the cell.
TextEdited	iGrid commits edit if the user edited the cell text, but it may remain the same when the edit finishes.
TextChanged	iGrid commits edit if the user edited the cell text and it changed.
ValueChanged	iGrid commits edit if the user changed the cell text and the value associated with it changed.
Never	iGrid never commits edit even if the user changed the cell.

The default value of the **iGrid.CommitEditCondition** property is **iGCommitEditCondition.Always**. This setting provides backward compatibility with the previous versions of iGrid when possible cell changes are committed for all actions that iGrid interprets as an attempt to save changes.

The **TextEdited** condition can help to track situations in which the user wants to force some updates even if the same text/value was entered. This condition differs from **Always** because in the case of **Always** the user may not change anything in the cell to commit edit. In the case of **TextEdited** the user must do something to change the cell text (for example, remove the last character and enter it again) to commit edit.

The difference between the **TextChanged** and **ValueChanged** conditions becomes clear if one value may have several text representations. As a rule, this is the case if you format numeric or date-time values with format strings. For example, if a cell contains numeric data, "100.0" and "100" entered into the cell will generate the same value of 100. In the case of **TextChanged** the edit will be committed, in the case of **ValueChanged** - not.

The **Never** condition can be used to implement a kind of read-only mode for the whole grid. If the **CommitEditCondition** property of iGrid is set to **Never**, the user will be able to put iGrid cells into edit mode to copy/paste cell texts or do anything with them, but any user changes will be always ignored when edit finishes.

2. [New] The event arguments of the **BeforeCommitEdit** and **AfterCommitEdit** events were supplemented with new fields to provide the same set of fields allowing you to know the values of the editable cell properties before and after editing. These are:

OldText	The previous text of the cell.
OldValue	The previous value of the cell.
OldImageIndex	The index of the previous image displayed in the cell.
OldDropDownControlItem	A reference to the drop-down control item selected in a combo box cell before editing.
NewText	The new text of the cell.
NewValue	The new value of the cell.
NewImageIndex	The index of the new image displayed in the cell.
NewDropDownControlItem	A reference to the drop-down control item selected in a combo box cell after editing.

3. [New] In Windows, if the user opens the drop-down list of a combo box cell, a click outside of the drop-down list closes the list without any subsequent action related to the mouse click. iGrid behaves the same way: if the user clicks outside of a drop-down control opened from a combo box cell, this action is typically treated as a command to close the drop-down control and do nothing more. This can lead to user misunderstanding if single-click edit mode is on: if the user opened a drop-down control in a cell and clicks another cell to activate editing in it, only the drop-down control will be closed and the user will need to click the other cell again to activate editing.

   The new Boolean **SingleClickEditOnDropDownClose** property of iGrid can help to solve this problem. Its value specifies whether iGrid will activate editing in the clicked cell if the user clicks outside of a drop-down control when single-click edit mode is enabled. The default value of this property is False that corresponds to the iGrid behavior described above. To enable editing with a single click when this click closes an opened drop-down control, set this property to True.

4. [New] The iGrid control provides 2 new methods to add or insert rows, **AddRow()** and **InsertRow():**

```
int AddRow()
int InsertRow(int rowBefore)
```

Note that these methods are methods of the **iGrid** control itself but not members of the **iGrid.Rows** collection. These new methods of iGrid return the index of the created row in contrast to their counterparts in the **iGrid.Rows** collection that return the **iGRow** object representing the created row. If you add a big number of rows using the new methods, you may find that their performance is about 7-8% higher compared to the methods of **iGrid.Rows** because a new **iGRow** object is not created after every method call.

But this is not the only benefit of the new methods. They have 2 overloaded versions that accept a variable number of arguments to initialize cell values in the created rows:

```
int AddRow(params object[] cellValues)
int InsertRow(int rowBefore, params object[] cellValues)
```

In the previous versions of iGrid, if you needed to create a new row and initialize its cells with some values, you wrote code like the following one:

```
iGRow myRow = iGrid1.Rows.Add();
myRow.Cells[0].Value = "Text";
myRow.Cells[1].Value = 100;
myRow.Cells[2].Value = 200;
```

Now you can simplify this code using the new **iGrid.AddRow()** method:

```
iGrid1.AddRow("Text", 100, 200);
```

The specified object values are used to initialize cells in columns with indices 0, 1, 2 and so on. If you specify more values than columns, the first of these additional values is used to set the row text cell value, the other values are ignored.

If you add a big number of rows and initialize cell values in them, the new **iGrid.AddRow()** method accepting cell values can run 50% faster compared to the traditional approach used in the previous versions.

5. [New] The **iGrid.Rows** collection provides 2 new methods for adding rows. They can be used to add/insert a given number of rows with the specified row pattern:

```
int AddRange(int count, iGRowPattern rowPattern)
int InsertRange(int rowBefore, int count, iGRowPattern rowPattern)
```

For example, the following code snippet adds five rows with a blue background to the end of iGrid:

```
iGRowPattern myRowPattern = new iGRowPattern();
myRowPattern.CellStyle = new iGCellStyle() { BackColor = Color.LightBlue };
iGrid1.Rows.AddRange(5, myRowPattern);
```

6. [New] AutoFilterManager provides an option to switch to the old way of populating the filter item list in the filter box from the main UI thread before displaying the filter box on the screen, which can cause delays in displaying the filter box for huge grids. This can be done with the new Boolean **BuildItemListInBackground** property of the AutoFilterManager component. Its default value is True and the filter box item list is built in the background thread not to block the UI. To use the old approach when this work is done in the main UI thread, set this property to False.

7. [Enhancement] The previous builds of PrintManager printed the texts of merged cells as bitmaps to provide the correct overall picture of merged cell parts if they should be printed on two or more pages.

As a result, texts of merged cells cannot be selected and copied to the clipboard if a grid had been printed to a PDF file.

The algorithm printing the contents of merged cells was rewritten in this update so that now texts of merged cells are printed as true texts and can be copied to the clipboard from PDFs. This also provides slightly better quality of merged cell texts compared to the previous output when they were printed as bitmaps.

Note that cell texts with additional formatting options applied, such as bold or italic, may still be saved as bitmaps in the resulting PDF files. This depends on the PDF printer driver and does not depend on PrintManager because this add-on always outputs cell texts as true texts now.

This improvement was done both for normal grid cells and for footer cells.

8. [Fixed] If the text of a merged cell was truncated by the cell edges and an ellipsis was displayed at the end of the text on the screen, PrintManager may have printed such a cell text without clipping and an ellipsis at the end.
9. [Fixed] Merged cells may have displayed incorrect grid lines if the developer changed the default settings for the grid lines drawn at the end of the frozen area and after the last row/column.
10. [Fixed] Incorrect cell contents may have been drawn in merged footer cells if some footer rows were hidden.
11. [Fixed] A ding sound was played when pressing ENTER to select an item from the drop-down list.
12. [Fixed] iGrid was not updated correctly if the user clicked another control while iGrid was in edit mode and the **Cancel** filed of the event arguments of the **BeforeCommitEdit** event was set to **iGEditResult.Cancel**.
13. [Fixed] Totals in footer rows were not updated correctly after clearing rows with the **iGrid.Rows.Clear()** method or after adding rows with the **iGrid.Rows.Add(iGRowPattern[] rowPatterns, iGCellPattern[] cellPatterns)**.

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.