10Tec iGrid for .NET 3.0
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;
- [Fixed] – a fixed bug or solved problem.

iGrid.NET Control Changes

1. [New] This build of iGrid fully supports .NET Framework 4.0 client profiles in Visual Studio 2010. The iGrid.NET control is implemented in two DLLs now, TenTec.Windows.iGridLib.iGrid.dll and TenTec.Windows.iGridLib.iGrid.Design.dll. The former is the core functionality which should be redistributed with your end-user applications. The latter is the implementation of the design-time features and is used only while you are developing your apps inside Visual Studio.

2. [New] Two new events, **CurCellChangeRequest** and **CurRowChangeRequest**, were implemented. They are raised when the user is about to change the current cell or row using the keyboard or mouse and deny this if required dynamically.

   In contrast to the **CurCellChanged** and **CurRowChanged** events which are raised after the current cell or row has been changed, the new events are raised before the change and allow you to deny the forthcoming change with the Boolean **DoDefault** parameter passed by reference. The other parameters of the new events are **RowIndex** and **ColIndex** (only for **CurCellChangeRequest**) which can be used to know the indices of the new row and column the user is going to make current.

   The **CurCellChangeRequest** and **CurRowChangeRequest** events are useful if you wish to protect some cells from selection dynamically in contrast to the **iGCell.Selectable** property, or when you need to perform some actions before the current cell or row is changed and abandon the change if something in your actions went wrong (for instance, you update the database when the user jumps from one row to another, and you need to stay in the current row if the update failed).

   Note that these events aren't triggered as a result of current cell or row change from your code using such methods as **SetCurCell** or **PerformAction**.

3. [New] iGrid implements two new events, **RowsAdded** and **RowsRemoved.** The first event is raised when a new row or a bunch of rows is added (or inserted) to the grid. The second event is raised when one row or a range of rows is removed. Both events have the **RowIndex** and **RowCount** parameters which are used to retrieve the index of the first inserted/removed row and their count (if one row is added/removed, **RowCount** equals 1).

4. [New] The **Header** object property has a new method called **Reset**. It removes all existing header rows except the first one and splits all merged column headers as if you had the header when you had just started to work with iGrid after defining some columns. All other parameters of column headers (width, text, color settings, etc) remain unchanged. This method can be useful in the situations when you need to change the set of header rows and/or column header parameters without changing the current cell set and such column properties as width and their order.

5. [New] The iGrid cell object (the **iGCell** class) has 3 new properties which return the effective cell type, cell type flags and the check box state for check box cells. These are the **EffectiveType**,

EffectiveTypeFlags and **EffectiveCheckState** properties of the **iGCellType**, **iGCellTypeFlags** and **System.Windows.Forms.CheckState** types respectively.

6. [New] A new static (shared in VB) **iGInternalInfrastructure** class was implemented. This class is designated to store the public members of iGrid.NET which are used only to implement internal functionality for interaction between the iGrid parts (such as its design-time features DLL or PrintManager add-on). The methods and types of this class are practically useless for the developer, and they are marked with the **EditorBrowsable** attribute with the **EditorBrowsableState.Never** setting so you'll never see them in the code editor in the IntelliSense member list. The **FillTreeBranchData** method of the iGrid class used by PrintManager was also moved into the **iGInternalInfrastructure** class in this version.

7. [New] A new **CustomDrawComboButton** event you can use to draw your own combo buttons was implemented. Its argument **iGCustomDrawComboButtonEventArgs** has the following fields:

```
public class iGCustomDrawComboButtonEventArgs : EventArgs
{
  public readonly int RowIndex;
  public readonly int ColIndex;
  public readonly Graphics Graphics;
  public readonly Rectangle Bounds;
  public readonly iGControlState State;
  public bool DoDefault;
}
```

   If you wish to redraw the combo button in a particular cell with the **RowIndex** and **ColIndex** indices, set the **DoDefault** parameter to False to prevent the default drawing and draw your own button in the **Bounds** rectangle taking into account the button state passed in the **State** field in this event. Note that the pressed combo button is always drawn as hot regardless whether the mouse pointer is inside its area or not, and the **State** filed reports the **iGControlState.HotPressed** in all these cases (**iGControlState.Pressed** is never used for combo buttons).

8. [Change] The default value of the **RowSelectionInCellMode** property was changed from 'MultipleRows' to 'None' which corresponds to the expectation of the users in the most cases.

9. [Change] The default height for rows and row normal cells was changed from 16 to 17 pixels. This allows to output the cell text using the default font with equal top and bottom margins in the cell and draw the cell combo buttons using OS visual styles better in Windows Vista and Windows 7.

10. [Renaming] The events **DrawEllipsisButtonBackground** and **DrawEllipsisButtonForeground**, the delegate used to define them **iGDrawEllipsisButtonEventHandler**, and the type of the event arguments **iGDrawEllipsisButtonEventArgs** were renamed to **CustomDrawEllipsisButtonBackground**, **CustomDrawEllipsisButtonForeground**, **iGCustomDrawEllipsisButtonEventHandler**, **iGCustomDrawEllipsisButtonEventArgs** respectively to conform to the general naming rule when all iGrid members related to custom drawing start with "CustomDraw".

11. [Change] The **e.State** argument of the **CustomDrawEllipsisButtonBackground** and **CustomDrawEllipsisButtonForeground** events always returns **iGControlState.HotPressed** for pressed buttons instead of **iGControlState.Pressed** in the previous versions as in iGrid the pressed ellipsis button is always drawn as hot regardless whether the mouse pointer is inside its area or not (the same behavior is used for iGrid combo buttons).

12. [Improved] The minimal size of the scroll bar thumb box was changed from 8 to 16, and this change allows to use the thumb box with the mouse easier when it has the minimal size.

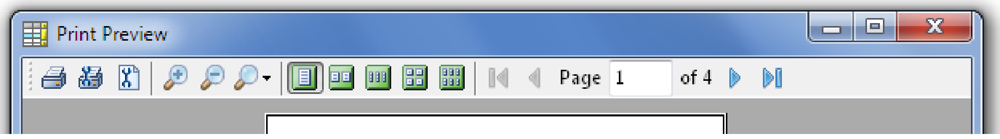
13. [Fixed] The thumb box on the scroll bars might be drawn incorrectly when it had the size of about 15 pixels and the OS visual style was used.

14. [Fixed] The drawing of ellipsis buttons was changed: now they are drawn exactly like command buttons, and their top part in rows of the default height isn't clipped when visual styles in Vista and Windows 7 are used as it was in the previous builds.

15. [Fixed] The previous builds of iGrid.NET might raise non-critical crash of Visual Studio 2010 while opening a C# project with iGrid.NET or you added iGrid.NET to a form in a C# project (you can still work with VS after closing the error dialog).

PrintManager Addon Changes

1. [Improved] The interface of the print preview window became more elegant – the main control toolbar was rewritten from scratch using the .NET ToolStrip control:



The detailed list of the toolbar changes is:

- Four new buttons, First Page, Previous Page, Next Page and Last Page, were implemented.
- A flat text box and label are used to display/enter the current page number and the total number of printed pages.
- The unneeded Close button was removed.
- Some superfluous separators were removed too.

2. [New] Two initial settings of the print preview window, window state and zoom factor, can be controlled from code using a new object property of PrintManager named **PrintPreviewSettings**. It has two nested properties for these purposes: **WindowState** and **Zoom**. The first of them is like the standard .NET form's **WindowState** property of the **System.Windows.Forms.FormWindowState** enumeration type. The second property accepts one of the values from a new **iGPrintPreviewZoom** enumeration corresponding the available zoom settings in the interface:

```
public enum iGPrintPreviewZoom
{
   Auto,
   Percent500,
   Percent200,
   Percent150,
   Percent100,
   Percent75,
   Percent50,
   Percent25
}
```

3. [New] PrintManager has a new **UIStrings** property which allows you to change all the strings you can see in the visual interface of PrintManager. It consists of two object subproperties, **PrintPreview** and **Warnings**, which are used to change the texts in the print preview window and user warnings. Here is an example of how you can set the title text for the print preview window:

```
iGPrintManager1.UIStrings.PrintPreview.Caption = "Annual Report";
```

All the string properties of the **UIStrings.PrintPreview** and **UIStrings.Warnings** object properties are marked with the .NET **Localizable** attribute, and thus the standard .NET localization technique can be used for them.

4. [New] Now you can use custom drawing to print your own headers and footers for the whole document or each page. You can draw your own contents in these bands in addition to the standard drawing or instead of it. These abilities allow you to implement such things as adding a logo to every page, using custom background (say, a gradient fill) for page bands, drawing additional graphics elements like lines, etc.

   Custom drawing is done in the following 4 new events: **CustomDrawDocumentHeader**, **CustomDrawDocumentFooter**, **CustomDrawPageHeader**, **CustomDrawPageFooter**. These events have the same signature and have the following parameters:

   - readonly Graphics **Graphics** – the Graphics object to draw on;
   - readonly Rectangle **Bounds** – the Rectangle structure to draw in;
   - readonly int **PageNumber** – the number of the current page;
   - readonly int **PageCount** – the total number of pages in the document;
   - bool **DoDefault** – a flag to tell PrintManager whether to draw the default contents.

   As you can see, the last parameter is passed by reference and allows you to prohibit the drawing of the default contents which can be performed after your custom drawing.

   The size and location of each band is determined based on the contents of such properties of PrintManager as **DocumentHeader**, **DocumentFooter**, **PageHeader** and **PageFooter**. You may need to change the bound of the required band if you wish to draw your own contents in addition to the specified text (for instance, you may print the current page using the **PageFooter.LeftSection.Text** property), or you may need to specify your own band placement if you do not use any text and wish to draw it from scratch. In this case, use the corresponding event from these 4 new events for this purpose: **CustomDrawDocumentHeaderGetBounds**, **CustomDrawDocumentFooterGetBounds**, **CustomDrawPageHeaderGetBounds**, **CustomDrawPageFooterGetBounds**. They all have the same following parameters:

   - readonly Graphics **Graphics** – the Graphics object to draw on;
   - Rectangle **Bounds** – the Rectangle structure to draw in;
   - readonly int **PageNumber** – the number of the current page;
   - readonly int **PageCount** – the total number of pages in the document;

   , and the **Bounds** parameter passed by reference is used to specify the required placement of the band. By default it is populated using the corresponding parameters of the printed page, but you can increase its height or even shift the left edge in these events.

   Pay attention to the fact that the page numeration (the **PageNumber** argument) starts from 0.

5. [Fixed] A serious problem with the drawing of the toolbar buttons in the print-preview dialog was fixed (they might be displayed without icons inside).
6. [Fixed] The Print… button might not work on some systems (nothing happened without any error message).
7. [Fixed] The Zoom In and Zoom Out buttons weren't disabled when they could not zoom in or zoom out the document anymore.

8. [Fixed] PrintManager ignored your page range settings when doing print-preview (these settings are accessed through the **PrintRange**, **MinimumPage**. **MaximumPage**, **FromPage**, **ToPage** properties of the **iGPrintManager.Document.PrinterSettings** object property).