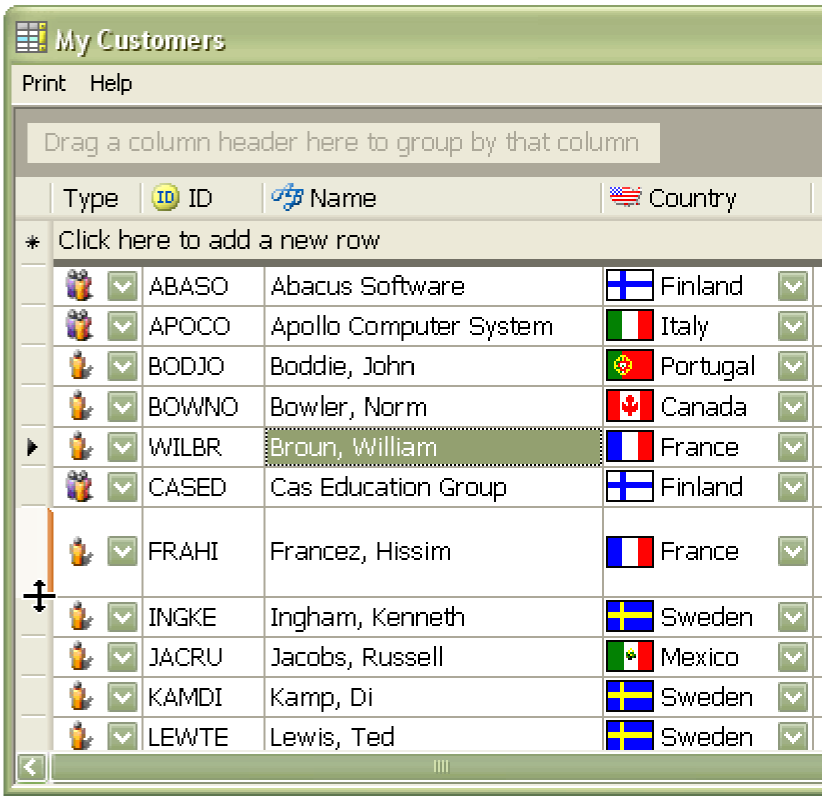
10Tec iGrid for .NET 2.0
What's New in the Control

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.

1. [New] Now iGrid.NET is compiled for the .NET Framework 2.0 and it can no longer be used with the previous versions of the .NET Framework.

2. [New] Now iGrid has a row header. It can be used to display row statuses (current, modified, etc) and to manipulate them (select/resize):


My Customers
Print
Help
Drag a column header here to group by that column
Click here to add a new row

Type	ID	Name	Country
	ABASO	Abacus Software	Finland
	APOCO	Apollo Computer System	Italy
	BODJO	Boddie, John	Portugal
	BOWNO	Bowler, Norm	Canada
	WILBR	Broun, William	France
	CASED	Cas Education Group	Finland
	FRAHI	Francez, Hissim	France
	INGKE	Ingham, Kenneth	Sweden
	JACRU	Jacobs, Russell	Mexico
	KAMDI	Kamp, Di	Sweden
	LEWTE	Lewis, Ted	Sweden

Most of the properties related to the row header are located in a new **RowHeader** object property of iGrid. This property returns an object of a new **iGRowHeader** class. The **iGRowHeader** class has the following properties:

- **Appearance** - determines the style of row header; acts only if the **UseXPStyles** property is set to False or the XP styles are turned off in the application or operating system.
- **BackColor** – determines the background color of the row header; if value of this property is set to False the **CellCtrlBackColor** property is used; this property does not affect the row header appearance when the XP styles are used.
- **DrawSystem** – determines whether to display the row header using system styles (3D, Flat, XP).
- **HGridLinesStyle** – the style of the row header's horizontal grid lines; use this property when the **DrawSystem** is set to False.

- **HotTracking** – determines whether the row header should indicate the hot state.
- **UseXPStyles** - determines whether to use the XP visual styles, if they are available, to display the row header; acts only if the **DrawSystem** property is set to True.
- **VGridLinesStyle** - the style of the row header's vertical grid lines; use this property when the **DrawSystem** is set to False.
- **Visible** – determines whether the row header is visible.
- **Width** – determines the width of the row header.
- **X** – Gets the X-coordinate of the row header.

By default iGrid's row header displays only a predefined set of glyphs: current row, editing, error, error in current row, new row, and uncommitted changes in current row (they all are defined in a new **iGRowHdrGlyph** enumeration). But you can display anything you want in it by using new custom draw events: **CustomDrawRowHdrBackground**, **CustomDrawRowHdrForeground**. These events have the **DoDefault** parameter which allows you to prohibit drawing of the default elements. If you draw custom contents in row headers, you should also handle new **CustomDrawRowHdrGetHeight** and **CustomDrawRowHdrGetWidth** events. These events are raised when a row height or row header width is automatically fit. With a new **DrawRowHdrGlyph** method, iGrid allows you to draw standard row header glyphs on a graphics surface.

If you want to determine the bounds of a separate row's header, you can use a new **HdrBounds** property of the row.

With a new **RowHdrDynamicBackColor** event you can dynamically specify the background color of a separate row header. By default its **Color** parameter is set to the background color of the whole row header; if you change it to a different color for a separate row header, the row header will be highlighted with this color.

If you want to automatically fit the row header width, you can invoke its new **AutoWidth** method.

Similar to cells and column headers, iGrid.NET provides you with new events which allow you to handle the mouse events when the mouse pointer is located over a row header. These events are: **RowHdrClick**, **RowHdrDoubleClick**, **RowHdrMouseDown**, **RowHdrMouseEnter**, **RowHdrMouseLeave**, **RowHdrMouseMove**, and **RowHdrMouseUp**.

3. [New] A new **HdrGlyph** property of the **iGRowHdrGlyph** enumeration was added to the row. This property is read-only. It returns a value indicating which glyph is currently displayed in the row's header.

4. [New] Now it is possible to resize the iGrid.NET rows through visual interface (using the row header), and you can control that with the following new properties: **RowResizeMode**, **ImmediateRowResizing**, and **AutoHeightRowMode**.

   The **RowResizeMode** property defines where resizing is enabled: in the row header, the frozen columns and row header, or all the columns and the row header.

   The **ImmediateRowResizing** property is similar to the **ImmediateColResizing** property and defines whether a row is resized immediately as the user drags its edge or it is resized when the user releases the left mouse button.

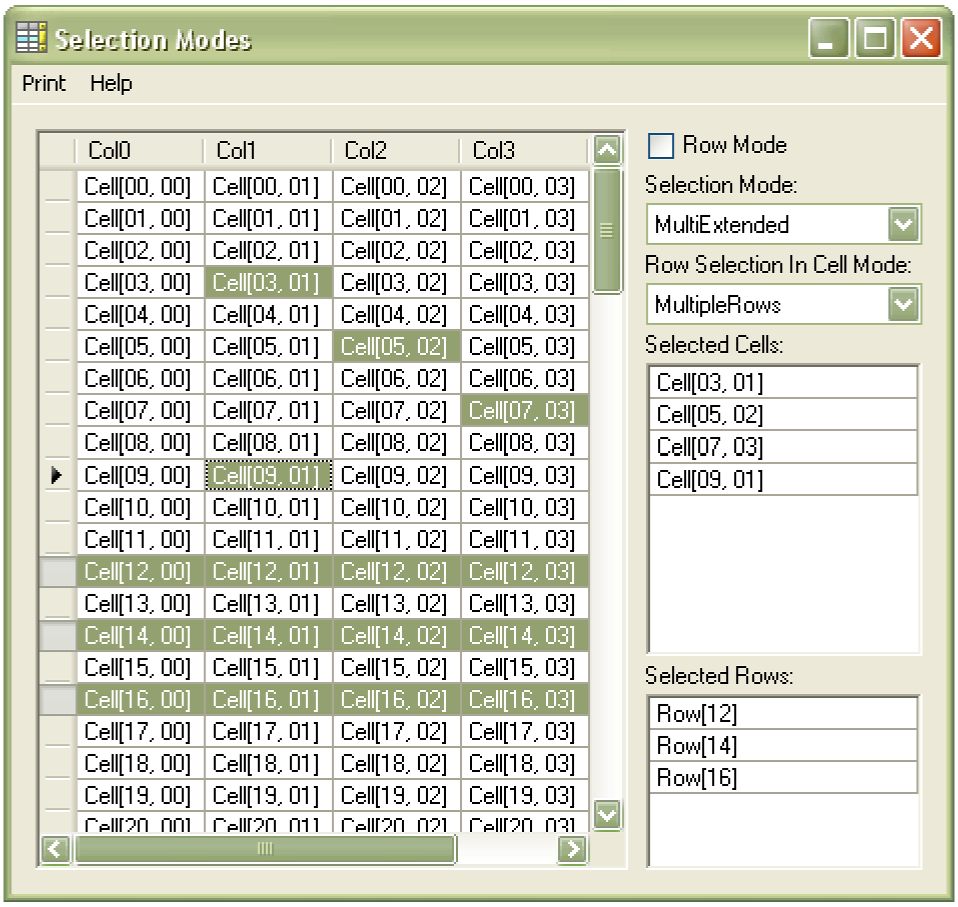
   The **AutoHeightRowMode** property indicates which parts of a row are taken into account when the user auto-heights the row either by double-clicking the row's edge in the area enabled for row resizing or when the row's height is fitted from code. If you want to catch the moment the user has double-clicked a row edge in the area enabled for row resizing or to prohibit auto-sizing a row, you can use a new **RowDividerDoubleClick** event.

To provide you with the information about the moments when the resizing of a row is performed, iGrid raises new **RowHeightStartChange**, **RowHeightChanging**, and **RowHeightEndChange** events before, during, and after you change the row height.

A new **RequestRowResize** event allows you to specify the minimal and maximal height and prohibit the resizing for individual rows.

5. [New] New **DrawRowHdrContents** and **DrawRowHeaderColHdrContents** methods were added to iGrid.NET. These methods allow you to draw the contents of a particular row header and the column header of the entire row header area on the specified graphics surface. The methods in fact are helper methods and are used mainly by the PrintManager add-on component.

6. [New][Change] Now iGrid maintains two different types of selection: row selection and cell selection. In the previous versions the selected cells (in cell mode) and rows (in row mode) were stored in a single collection which you were able to access with the **SelectedCells** property of iGrid. Now iGrid has two properties, **SelectedCells** and **SelectedRows**, which store the selected cells and rows respectively. This gives the user the ability to select rows in cell mode:



Col0	Col1	Col2	Col3
Cell[00, 00]	Cell[00, 01]	Cell[00, 02]	Cell[00, 03]
Cell[01, 00]	Cell[01, 01]	Cell[01, 02]	Cell[01, 03]
Cell[02, 00]	Cell[02, 01]	Cell[02, 02]	Cell[02, 03]
Cell[03, 00]	Cell[03, 01]	Cell[03, 02]	Cell[03, 03]
Cell[04, 00]	Cell[04, 01]	Cell[04, 02]	Cell[04, 03]
Cell[05, 00]	Cell[05, 01]	Cell[05, 02]	Cell[05, 03]
Cell[06, 00]	Cell[06, 01]	Cell[06, 02]	Cell[06, 03]
Cell[07, 00]	Cell[07, 01]	Cell[07, 02]	Cell[07, 03]
Cell[08, 00]	Cell[08, 01]	Cell[08, 02]	Cell[08, 03]
Cell[09, 00]	Cell[09, 01]	Cell[09, 02]	Cell[09, 03]
Cell[10, 00]	Cell[10, 01]	Cell[10, 02]	Cell[10, 03]
Cell[11, 00]	Cell[11, 01]	Cell[11, 02]	Cell[11, 03]
Cell[12, 00]	Cell[12, 01]	Cell[12, 02]	Cell[12, 03]
Cell[13, 00]	Cell[13, 01]	Cell[13, 02]	Cell[13, 03]
Cell[14, 00]	Cell[14, 01]	Cell[14, 02]	Cell[14, 03]
Cell[15, 00]	Cell[15, 01]	Cell[15, 02]	Cell[15, 03]
Cell[16, 00]	Cell[16, 01]	Cell[16, 02]	Cell[16, 03]
Cell[17, 00]	Cell[17, 01]	Cell[17, 02]	Cell[17, 03]
Cell[18, 00]	Cell[18, 01]	Cell[18, 02]	Cell[18, 03]
Cell[19, 00]	Cell[19, 01]	Cell[19, 02]	Cell[19, 03]
Cell[20, 00]	Cell[20, 01]	Cell[20, 02]	Cell[20, 03]

☐ Row Mode

Selection Mode: MultiExtended

Row Selection In Cell Mode: MultipleRows

Selected Cells:
Cell[03, 01]
Cell[05, 02]
Cell[07, 03]
Cell[09, 01]

Selected Rows:
Row[12]
Row[14]
Row[16]

In cell mode rows can be selected by using the row header (click a row header to select the row). To select several rows, hold the Control modifier key and consequently click the headers of the rows to be selected. You can also use the Shift modified key to select a group of rows located consequently: hold the Shift key, click the first row to select and then click the last row to select. A new property named **RowSelectionInCellMode** allows you to specify how many rows can be selected in cell mode. Whereas the old **SelectionMode** property works as it was before: in cell mode it defines the cell selection mode; in row mode, the row selection mode.

A new **Selectable** property of the row allows you to mark rows as non-selectable.

A new **Selected** property of the row allows you to select/deselect a separate row and determine whether the row is selected.

The **HighlightBackColor**, **HighlightForeColor**, **HighlightBackColorNoFocus**, and **HighlightForeColorNoFocus** properties were removed. Instead of them new properties, which separately define the selected cell and row colors, were added: **SelCellsBackColor**, **SelCellsForeColor**, **SelCellsBackColorNoFocus**, **SelCellsForeColorNoFocus**, **SelRowsBackColor**, **SelRowsForeColor**, **SelRowsBackColorNoFocus**, and **SelRowsForeColorNoFocus**. By default the selected row colors are set to the Empty value. It means that the cell selection color is used. The selection background colors can be set to semi-transparent colors. It allows you to make it possible to see which cells in selected rows are selected as well as to see the background color of the selected cells.

A new **HighlightSelCellItems** property was added to the grid. It allows you to specify which cell parts (controls and images) should be highlighted when the cell or row it is located in is selected.

Now the iGrid.NET cell is drawn in the following sequence:

a. The grid's background is drawn.

b. The cell is filled with its original background color.

c. If the cell is located in a selected row, and **SelRowsBackColor** is opaque, the cell is filled with **SelRowsBackColor**. If **SelRowsBackColor** is not opaque but a cell part (the image or controls) should not be highlighted, this part is filled with **SelRowsBackColor**.

d. If the cell is selected, and **SelCellsBackColor** is opaque, the cell is filled with **SelCellsBackColor**. If **SelCellsBackColor** is not opaque but a cell part (the image or controls) should not be highlighted, this part is filled with **SelCellsBackColor**.

e. If the cell is current and **CurCellBackColor** is opaque and not empty, the cell is filled with **CurCellBackColor**. If **CurCellBackColor** is not opaque but a cell part (the image or controls) should not be highlighted, this part is filled with **CurCellBackColor**.

f. The cell image and controls are drawn.

g. If the cell is located in a selected row, and a cell part (the image or controls) should be highlighted (controlled with the new **HighlightSelCellItems** property), the part is filled with **SelRowsBackColor**. If **SelRowsBackColor** is opaque, its semi-transparent equivalent is used instead.

h. If the cell is selected, and a cell part (the image or controls) should be highlighted, the part is filled with **SelCellsBackColor**. If **SelCellsBackColor** is opaque its semi-transparent equivalent is used instead.

i. If the cell is current, and a cell part (the image or controls) should be highlighted, the part is filled with **CurCellBackColor**. If **CurCellBackColor** is opaque its semi-transparent equivalent is used instead.

j. The cell text is drawn.

Now you can set the selected cell and row foreground color to the Empty value. If you do so, the selected cell text will be drawn with the normal cell foreground color.

Previously you used the universal **SelectAll** and **DeselectAll** actions (the **iGActions** enumeration) to select or deselect all the cells or rows in row mode. Now these actions are separated into four new ones: **SelectAllCells, DeselectAllCells, SelectAllRows,** and **DeselectAllRows**. So now you can separately select or deselect all the rows or cells in iGrid.

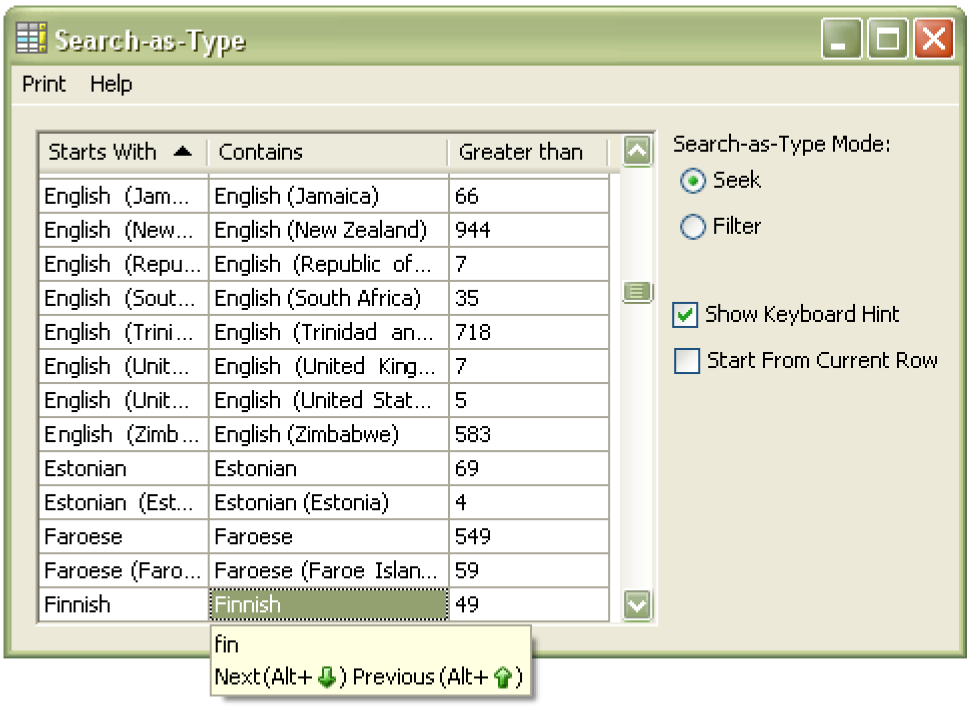
As a result of the changes listed above, the **CustomDrawCellBackground** and **CustomDrawCellForeground** events now have one additional argument named **RowSelected** which indicates whether the cell being drawn is in a selected row.

7. [New][Change] iGrid displays some standard strings in its interface items (such as the default label "Drag a column header here to group by that column" in its group box), and you may need to change those strings in localized versions of your apps. A new **UIStrings** object property was introduced in iGrid.NET for that purpose. It returns an object which properties allow you to change all the standard strings used in the user interface of iGrid.

   As a result, the **Text** property was removed from the **iGGroupBox** class which instance is returned by the **GroupBox** property of iGrid. Now if you want to change the text which is displayed in the group box when no columns are grouped, you should use the **GroupBoxHintText** property of the **UIStrings** object property of iGrid.

   The **UIStrings** property is marked with the localizable attribute, which means that you can specify different strings for different languages by using the Windows Forms Designer.

8. [New] Search-as-type functionality was added to iGrid. This functionality (also known as incremental search) allows you to find the necessary cell by typing characters the cell's text starts with, contains, etc while the grid is focused:



Starts With ▲	Contains	Greater than
English (Jam...	English (Jamaica)	66
English (New...	English (New Zealand)	944
English (Repu...	English (Republic of...	7
English (Sout...	English (South Africa)	35
English (Trini...	English (Trinidad an...	718
English (Unit...	English (United King...	7
English (Unit...	English (United Stat...	5
English (Zimb...	English (Zimbabwe)	583
Estonian	Estonian	69
Estonian (Est...	Estonian (Estonia)	4
Faroese	Faroese	549
Faroese (Faro...	Faroese (Faroe Islan...	59
Finnish	Finnish	49

fin
Next(Alt+⇩) Previous(Alt+⇧)

Search-as-Type Mode:
[✓ Seek]
Filter

[✓] Show Keyboard Hint
[ ] Start From Current Row

A new **SearchAsType** object property returns an object of the **iGSearchAsType** type which exposes all the properties related to the search-as-type functionality.

Searching can work in two modes: seek and filter. In the first one iGrid moves the current cell to the position which matches the search criteria as the user is typing characters. In the second mode iGrid not only moves the current cell, but also hides all the rows that do not match the search criteria. The property which allows you to switch between these modes is the **Mode** property of the **iGSearchAsType** type.

The criteria which iGrid uses to determine which cells match the string being sought are specified with the **MatchRule** property of the **iGSearchAsType** type. This property accepts one of the following values:

- **StartsWith** – the cell text should start with the entered text.
- **Contains** – the cell text should contain the entered text.
- **Custom** – the **Match** argument of a new **SearchAsTypeCustomCompare** event should be set to True.

When the **Custom** value is specified, iGrid raises the **SearchAsTypeCustomCompare** event when it needs to determine whether a cell matches the search criteria. If the cell matches, you should set its **Match** parameter to True. Note that as this type of search is incremental, the number of rows which match the search criteria should not be increased as the user is typing new characters when filtering. In other case the functionality will work incorrectly as optimizing algorithms are used.

By default a single match rule is used for all the columns, but you can change this behavior by handling a new **RequestSearchAsTypeMatchRule** event. It is raised every time searching as typing is started. Its **MatchRule** argument allows you to dynamically specify a match rule for a column.

A new **SearchAsTypeRowSetChanged** event was added to iGrid, it occurs after the set of visible rows has changed when searching as typing.

By default iGrid uses for searching the column which contains the current cell. But you can change this behavior by changing the **SearchCol** property of the **iGSearchAsType** type. When a column is attached to this property, the search-as-type functionality is enabled, the grid is focused, and the user enters a character, the current cell is moved to the first cell in the search column which matches the search criteria or/and filters the rows. To support this property at design-time, a new **FromIndex** method was added to the column collection. It returns a column by its index.

The Boolean **StartFromCurRow** property of the **iGSearchAsType** type allows you to specify whether to start searching from the current row or from the beginning of the grid.

The Boolean **LoopSearch** property of the **iGSearchAsType** type tells whether to continue searching from the beginning of the grid when the end is reached.

When the search-as-type functionality is active, iGrid displays a special window, which is named search window, where the search text is shown. By default, this window also displays special keyboard hints with the keyboard combinations which allow the user to move to the next (Alt + Down Arrow) and previous (Alt + Up Arrow) records matching the search criteria. The Boolean **DisplaySearchText** and **DisplayKeyboardHint** properties of the **iGSearchAsType** type allow you to specify which parts of the search window to display. If both of these properties are set to False, the search window is not displayed at all.

The standard text of the keyboard hint in the search window (the 'Next' and 'Previous' strings) can be changed or translated into another language. To do it, modify the **SearchWindowLabelNext** and **SearchWindowLabelPrev** properties of the **UIStrings** object property of iGrid.

The search-as-type functionality supports the following key combinations:

- Alt + Down Arrow – go to the next matching cell.
- Alt + Up Arrow – go to the previous matching cell.
- Esc – cancel searching as typing.
- Backspace – remove the last character from the search string.

When the user enters a character after which there are no matches in the grid, this character is displayed red in the search window and an exclamation icon is displayed on the right (left in right-to-left mode) of it. After that if you enter a new character, it will replace the previous one in the search string (when the current search string has matches in the grid, characters are added to the search string). So you will never write more than one invalid character.

In addition to the standard use of the search-as-type functionality, you can use it for your specific needs, for example search by a text entered into another control such as a text box. For this purpose **iGSearchAsType** type exposes the following properties and methods:

- The **AutoCancel** property – indicates whether to automatically cancel searching as typing when the grid loses the focus, the form the grid is located on is deactivated etc. When this property is set to False, the searching is cancelled only when the **Cancel** method of the **SearchAsType** object property is invoked, or the search text is set to the empty string. If you set this property to False, take into account that the search window is not hidden until you cancel searching manually.
- The **Cancel** method – cancels searching as typing and clears the search string.
- The **FilterKeepCurRow** property – if set to True, the current row is not changed when in filter mode.
- The **GoNext** method – moves the current cell to the next cell which matches the search criteria.
- The **GoPrev** method – moves the current cell to the previous cell which matches the search criteria.
- The **HasMatches** property – indicates whether the currently specified search text has matches in the grid.
- The **IsActive** property – indicates whether the searching as typing is currently active.
- The **SearchText** property - gets or sets the current search text. When this property is modified, iGrid automatically moves the current cell to the matching one and/or filter the rows.

9. [New][Change] iGrid.NET 2.0 no longer has two different cell types – text cell and combo cell, they are combined into one text cell which can have an attached drop-down list and optionally the combo button. Due to this change the **Combo** member of the **iGCellType** enumeration was removed. Now when you want to create a combo cell, it is enough to assign a drop-down control to the cell (using the **DropDownControl** property) – the combo button will be displayed automatically. Here is an example of how to create combo cells in the first column:

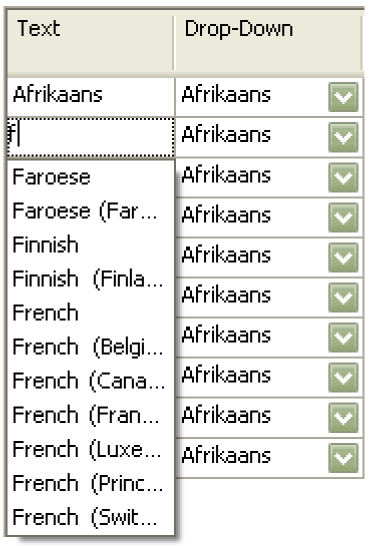
```
iGDropDownList myList = new iGDropDownList();

myList.Items.Add("Value1");
myList.Items.Add("Value2");
myList.Items.Add("Value3");

iGrid1.Cols[0].CellStyle.DropDownControl = myList;
```

A new **HideComboBtn** flag was added to the **iGCellTypeFlags** enumeration. This flag allows you to hide the combo button in a cell when a drop-down control is attached to it.

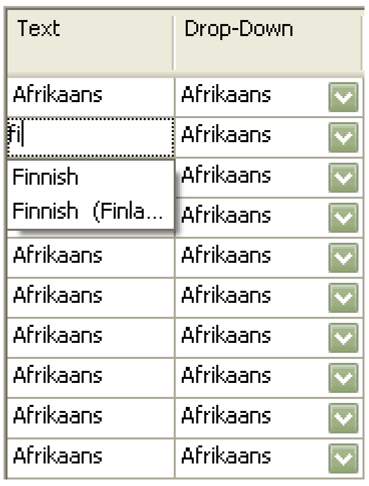
The **ComboEditable** flag was also removed from the **iGCellTypeFlags** enumeration. Now when a drop-down control is attached to a cell with the **DropDownControl** property of its style, the cell automatically becomes editable as text and with drop-down control. If you want to prohibit editing cell as text, you should use a new **NoTextEdit** flag. This flag should be assigned to the **TypeFlags** property of the cell style.

10. [New] Auto-complete functionality was added to iGrid. It helps you to enter a value to a cell when you use the keyboard and provides you with a list of possible values.

    The auto-complete functionality starts to work when the user is editing a cell as text, and an auto-complete list is attached to this cell. If the auto-complete list has values which correspond to the entered text (values start with the text, contain the text etc), they are displayed, and the user can choose one of them:



Text	Drop-Down
Afrikaans	Afrikaans
f	Afrikaans
	Afrikaans
	Afrikaans
	Afrikaans
	Afrikaans
	Afrikaans
	Afrikaans
	Afrikaans
	Afrikaans



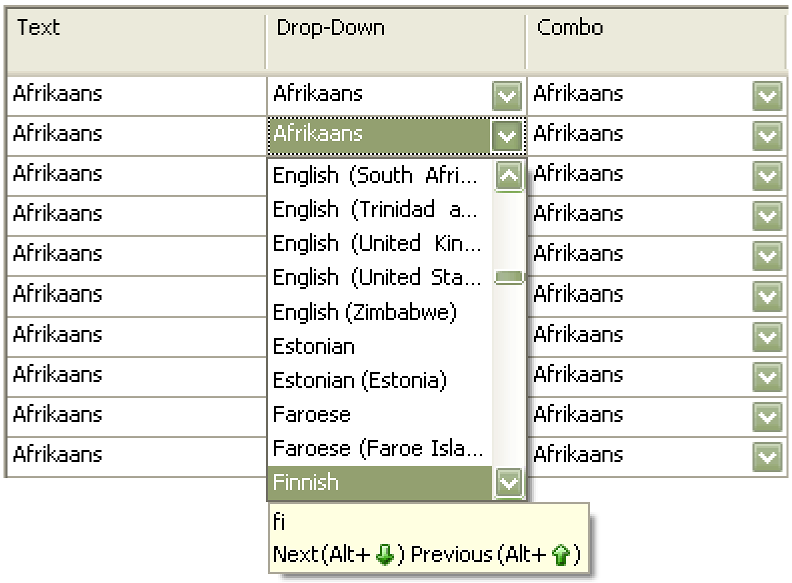
Text	Drop-Down
Afrikaans	Afrikaans
fi	Afrikaans
	Afrikaans
	Afrikaans
Afrikaans	Afrikaans
Afrikaans	Afrikaans
Afrikaans	Afrikaans
Afrikaans	Afrikaans
Afrikaans	Afrikaans
Afrikaans	Afrikaans

    The auto-complete list is assigned to a cell with the **DropDownControl** property of a style attached to the cell (by default the auto-complete control is the control which also appears when you start editing a cell by clicking the combo button or pressing the F4 key). To be used for the auto-complete functionality, the object assigned to the **DropDownControl** property should implement a new **IiGAutoCompleteControl** interface. The standard drop-down lists are based on the **iGDropDownList** class which now implements the **IiGAutoCompleteControl** interface and thus can be used as an auto-complete control. When you attach it to a cell, it will be shown when the user clicks the combo button, presses the F4 key, or is editing the cell as text.

    Now **iGDropDownList** (i.e. the standard drop-down list control) can work in two modes: auto-complete and drop-down. When it works in auto-complete mode, its rows can be either filtered based on the text currently being entered to a cell (filter mode) or the first matching item may be highlighted (seek mode). This behavior is specified with the **AutoCompleteMode** property of a new **SearchAsType** object property of the **iGDropDownList** class. When the user enters a text into a cell, and an auto-complete list is shown in seek mode, the following key combinations can be used:

    - Enter – save the value selected in the auto-complete list.
    - Shift + Enter – save the text entered to the cell.
    - Esc – cancel editing without saving changes.
    - Alt + Up Arrow/Down Arrow – go to the next/previous matching item in the auto-complete list.

    When **iGDropDownList** works in drop-down mode, you can also use the search-as-type functionality and filter or seek the required item. The **DropDownMode** property of the **SearchAsType** object property of the **iGDropDownList** class allows you to switch between filtering and seeking. The control keys in this case are the same as in iGrid when searching as typing:



Text	Drop-Down	Combo
Afrikaans	Afrikaans	Afrikaans
Afrikaans	Afrikaans	Afrikaans
Afrikaans	English (South Afri…	Afrikaans
Afrikaans	English (Trinidad a…	Afrikaans
Afrikaans	English (United Kin…	Afrikaans
Afrikaans	English (United Sta…	Afrikaans
Afrikaans	English (Zimbabwe)	Afrikaans
Afrikaans	Estonian	Afrikaans
Afrikaans	Estonian (Estonia)	Afrikaans
Afrikaans	Faroese	Afrikaans
	Faroese (Faroe Isla…	
	Finnish	

fi
Next(Alt+⇩) Previous(Alt+⇧)

The criteria used in the search-as-type functionality of **iGDropDownList** (in both drop-down and auto-complete modes) are specified with the **MatchRule** property of the **SearchAsType** object property of **iGDropDownList**. This property accepts the same values as the **MatchRule** property of the **SearchAsType** object property of iGrid. When the **Custom** value is specified, a new **SearchAsTypeCustomCompare** event of **iGDropDownList** is used to determine whether an item matches the search criteria.

The same search window, which displays the search text and keyboards hint, as in iGrid is also displayed for **iGDropDownList**. But whether each part of the search window is displayed depends on the context which the search window is shown in:

- Drop-down control in seek mode – both parts are displayed.
- Drop-down control in filter mode – only the search text is displayed.
- Auto-complete control in seek mode – only the keyboard hint is displayed.
- Auto-complete control in filter mode – the search window is not displayed at all.

In addition **iGDropDownList** allows you to prohibit showing the search window parts. It can be done with the **DisplayKeyboardHintIfNeeded** and **DisplaySearchTextIfNeeded** properties.

In some cases it is needed that the auto-complete control (which is shown when editing a cell as text) should differ from the drop-down control (which is shown when the user clicks the combo button or presses the F4 key). For example, you use a custom cell drop-down editor which does not support the auto-complete functionality. In this case you should use a new **RequestAutoCompleteControl** event of iGrid. This event is raised every time iGrid needs to obtain the auto-complete control for a cell. Just set the **Control** argument of this event to your auto-complete control, and that's all.

A new **RequestDropDownControl** event was also added to iGrid. It is similar to the **RequestAutoCompleteControl** event, and is raised every time iGrid needs to obtain the drop-down control for a cell. In this event you can dynamically attach a drop-down control to a cell.

In order to support auto-complete functionality, the **GetDropDownControl** method of the **IiGDropDownControl** interface was changed: a new **interfaceType** parameter was added to it. This parameter informs the implementer which interface the method is invoked for (this parameter can be the type of **IiGDropDownControl** itself or the type of any derived interface such as **IiGAutoCompleteControl**).

Also two new methods, **OnShow** and **OnHide**, were added to the **IiGDropDownControl** interface. These methods are invoked when the drop-down control is shown and hidden respectively.

In addition to the standard implementation of the auto-complete functionality (**iGDropDownList**), you can create your own auto-complete control. To do it, you should implement the **IiGAutoCompleteControl** interface. This interface is derived from the **IiGDropDownControl** interface and has the following members:

- The **OnCellTextChange** method – this method is invoked by iGrid when the text in a cell is changed while editing this cell as text. Its only parameter, which is named text, provides you with the text entered to the cell. This method should return a Boolean value indicating whether the auto-complete control has items which match the entered text and it should be displayed.
- The **ProcessKeyDown** method - processes the key down event when a cell is being edited as text. This method is invoked when the control which is used to edit the cell (a text box) receives the KeyDown event. The arguments of this event are passed in the **e** parameter of the method. If the auto-complete control has handled this event and wants to prevent the cell editor from handling the key, you should set the **Handled** parameter of the event arguments to True.
- The **ProcessKeyPress** method - processes the key press event when a cell is being edited as text. This method is invoked when the control which is used to edit the cell (a text box) receives the KeyPress event. The arguments of this event are passed in the **e** parameter of the method. If the auto-complete control has handled this event and wants to prevent the cell editor from handling the key, you should set the **Handled** parameter of the event arguments to True.
- The **ProcessKeyUp** method - processes the key up event when a cell is being edited as text. This method is invoked when the control which is used to edit the cell (a text box) receives the **KeyUp** event. The arguments of this event are passed in the **e** parameter of the method. If the auto-complete control has handled this event and wants to prevent the cell editor from handling the key, you should set the **Handled** parameter of the event arguments to True.
- The **ValueSelected** event – should be raised when a value in the auto-complete control is selected and editing should be finished.

11. [New] A new **UpdateDropDownLocationAndSize** method was added to iGrid. Invoke this method if you implement a custom drop-down control which contents (and a result size) are changing during editing. It causes iGrid to adjust the size and location of the drop-down control which is currently shown in a cell.

12. [New] A new **ProcessEnter** property was added to iGrid. This property determines whether iGrid should process the Enter key and start editing of the current cell. Earlier iGrid always processed the Enter key, and, as a result, if iGrid was shown in a dialog box with an accept button, the Enter key did not close the dialog.

13. [New] A new **CellDynamicContents** event was added to iGrid. This event occurs every time when iGrid needs to obtain the text and image which are displayed in a cell. You can specify the cell contents dynamically by using this event. For instance, you can store numeric values in cells and convert them on the fly to their corresponding string representations on the screen in this event.

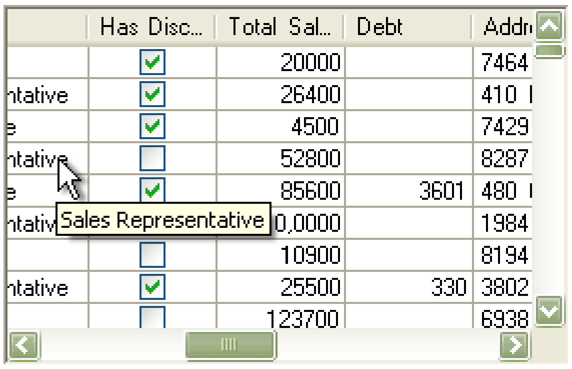
14. [New][Change] A new **CellDynamicFormatting** event was added to iGrid. This event occurs every time when iGrid needs the background or foreground color or font of a cell, and this event allows you to specify these cell parameters dynamically based on custom conditions. The old methods named

DynamicBackColor, **DynamicForeColor**, and **DynamicFont** were removed from iGrid - now if you change several of these parameters dynamically using the same logic, you can do that in one event instead of several events in the previous version.

15. [Change] The **DynamicStringFormat** event was renamed to **CellDynamicStringFormat**.

16. [New] Now when a cell text is not entirely displayed (clipped) in a cell or column header, a tool tip is shown. You can show your own text in this tool tip by handling new **RequestCellToolTipText** and **RequestColHdrToolTipText** events (assign the required string to their **Text** argument). If you do not want the tool tip to be displayed, set the **Text** argument to null (Nothing in VB).

    Note that for better user experience, the tool tip is shown not only when the cell text is clipped at the end (when the ellipsis character is displayed) but also when the cell text is hidden by another grid element – for instance, under the vertical scroll bar, or say the height of the row is not enough to display the cell text without clipping:



	Has Disc...	Total Sal...	Debt	Addr
	✓	20000		7464
ntative	✓	26400		410 I
e	✓	4500		7429
ntative	☐	52800		8287
e	✓	85600	3601	480 I
ntativ		0,0000		1984
	☐	10900		8194
ntative	✓	25500	330	3802
	☐	123700		6938

Sales Representative

    The tool tip is also displayed for the check box cells as well in the same situations.

17. [New] A new **IsCellPartClipped** method was added to the **iGCell** class. This method allows you to determine whether one of the following cell parts - image, text, or check box - is clipped. The **parts** parameter of this method determines the cell parts which should be checked. The **checkIfPartiallyHidden** parameter allows you to specify whether this method should check if the specified cell parts are partially or fully hidden by a cells area edge, i. e. are out of the viewport.

    In fact, this method wraps the internal functionality iGrid uses to determine whether the cell contents is clipped and the default tool tip should be displayed, and you can also use it in your programs to know whether a cell part is clipped and it may require a tool tip to display its full contents.

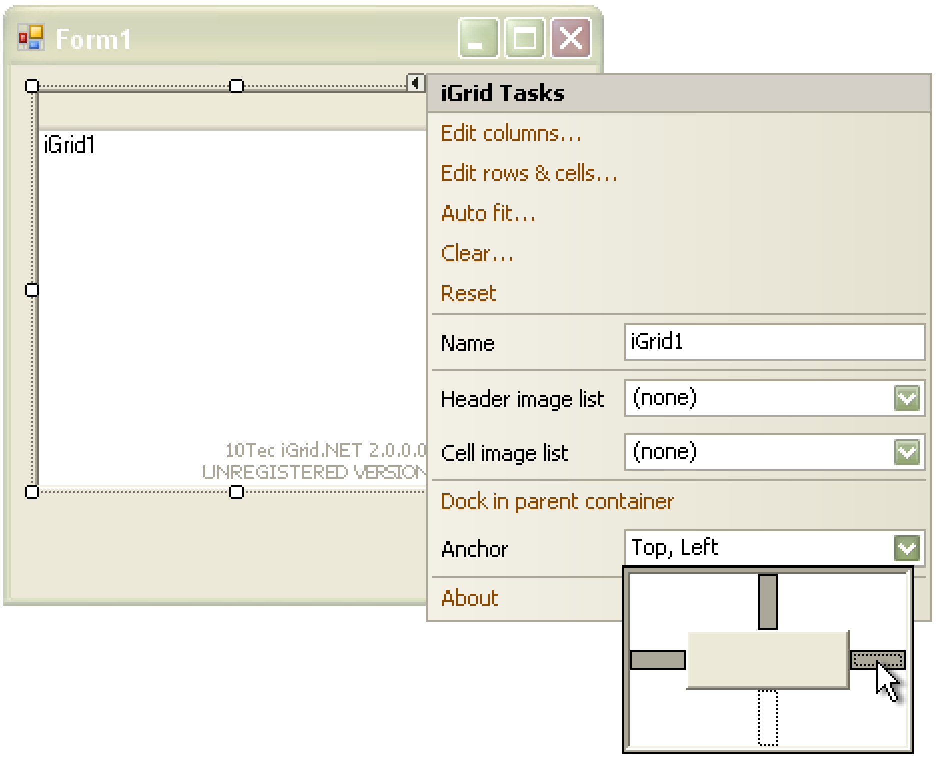
18. [New] A new **RequestPageBreak** event was added to the Print Manager add-on component. This event allows you to break pages after/before particular rows when printing.

19. [New][Change] Now when you invoke the **AutoWidth** method of the row text column (the index of this column equals -1), the grid adjusts both the width of the row text cells (which are located under the normal cells) and the width of all the columns to fit the contents of the group rows. In the previous versions only the row text cells were adjusted.

20. [New] New **KeyExists** methods were added to the column and row collections. These methods return a boolean value indicating whether the specified key is defined on the column or row collection.

21. [New] A new boolean **ColorizeRowLevelIndent** property was added. This property determines whether to fill the indent level empty area of a row with the row style's background color and the selection background color if it is selected.

22. [New] You can notice new enhancements when you work with iGrid in the Windows Forms Designer. First, new designer actions were added to iGrid. They allow you to set the name of the control, choose its image lists, dock it in the parent container and anchor the control's edges. Second, the name of a grid is displayed in the left upper corner of the cells area when the grid has no cells. It is a very useful feature when you have several grids on your form - now you don't need to remember the name of each grid:



Name: iGrid1
Header image list: (none)
Cell image list: (none)
Anchor: Top, Left

23. [Change] Now when you set the **Visibility** property of a scroll bar to the **Hide** value, the user still can scroll the grid by pressing the arrow keys or by using the mouse wheel. If you want to prevent iGrid from scrolling, you should set the **Enabled** property of its scroll bars to False; you can do it for each scroll bar individually depending on the scroll direction you want to prohibit.

24. [Change] All the members which were marked as obsolete in the previous versions were removed.

25. [Change] Now iGrid does not handle its internal errors itself. This was done for iGrid to comply with common Windows Vista rules for developing applications and components. As a result the **DebugMode** property was removed.

26. [Change] In the previous versions when a cell was smaller than the default size of the combo or ellipsis button, these buttons were scaled to fit the cell. When scaling, the buttons preserved their aspect ratio. Now the buttons are scaled independently vertically and horizontally. For example, if a cell's height is smaller than the combo button default height, iGrid adjusts the height of the button to fit the cell's height, but the width of the button is preserved (if the cell is wide enough).

27. [Fixed] When a button, which was located on the same form with iGrid, had a hot key, and this hot key was pressed when the input focus was on iGrid, the **Click** event of the button was raised, and editing in iGrid was not started.

28. [Fixed] When the grid is in the single-click edit mode, the cells with the **Selectable** property set to False were still editable.

29. [Fixed] When one iGrid was being edited, and the user clicked another iGrid, the cell in the other grid was selected and the focus was moved to the first grid.

30. [Fixed] When the last visible row's horizontal grid line was thinner than the normal horizontal grid line, the level area was filled incorrectly.

31. [Fixed] When a column header had a very small width, the user could not click it to sort the column because the resize cursor was displayed all the time when the mouse pointer was over the column header.

32. [Fixed] When a cell had a semi-transparent background color, it could not be edited as text.

33. [Fixed] In some cases the **Bounds** argument of the column header mouse events and the **Bounds** property of the column header returned wrong values.