10Tec iGrid ActiveX 6.x
What's New in the Latest Builds

Keywords used to classify changes:

- [New] – a totally new feature;
- [Change] – a change in a member functionality or interactive behavior;
- [Fixed] – a fixed bug or solved problem;
- [Removed] – a member was completely removed;
- [Enhancement] – some functionality was enhanced;
- [Optimization] – a feature has speed improvements;
- [Renaming] – a member was renamed;
- [Code-Upgrade] – a special tag used to mark changes that may require changes in the source code for older versions.

v6.50, build 0080 | 2017-Sep-28

1. [New] A new experimental feature that allows you to set the maximal number of rows processed when iGrid calculates the optimal width for a column(s) in the **AutoWidthCol**/**AutoWidthCols** methods or when the user double-clicks column dividers has been implemented. To enable this feature, call the **DevDbg** method with the parameter index 4 and specify the required number of records to process as the parameter value. For example, the following call limits the number of processed rows to the first 100 rows in the grid:

   ```
   iGrid1.DevDbg 4, 100
   ```

   This call affects all subsequent auto-width-column operations. To return to the default behavior, when all grid rows are processed, specify 0 as the parameter value in the **DevDbg** call:

   ```
   iGrid1.DevDbg 4, 0
   ```

   This experimental feature is a subject to change in the future builds of iGrid and, most likely, it will be implemented as a new member of iGrid in the next major/minor version update.

2. [New] A new experimental feature that allows you to specify the background color of the active column header when visual styles in the header are turned off has been implemented. To specify the required hot-track background color, call the **DevDbg** method with the parameter index 5 and specify the color as the parameter value. For example, the call below makes the background color of the column header under the mouse pointer red:

   ```
   iGrid1.DevDbg 5, vbRed
   ```

   To reset this hot-track background color to the default empty value, specify CLR_NONE (-1) as the parameter value:

   ```
   iGrid1.DevDbg 5, -1
   ```

   This experimental feature is a subject to change in the future builds of iGrid and, most likely, it will be implemented as a new member of iGrid in the next major/minor version update.

3. [Change] The auto-scrolling feature available while the user is dragging column headers to reorder columns has been disabled for the case when iGrid has frozen columns. This is done to avoid side effects that lead to improper column reordering because of some limitations of the native Microsoft Header control used as the container for iGrid column headers.

4. [Fixed] The horizontal scrolling performed when the user pressed the LEFT ARROW key worked improperly if the grid had frozen columns, some non-frozen columns were invisible and the **BrowseAllNonFrozenCols** property was set to True.
5. [Fixed] The CTRL+SHIFT+C keyboard combination to copy the selected grid contents with the column headers did not work.
6. [Fixed] iGrid crashed when the user tried to open a combo box with a non-existing key (for example, if the developer assigned a key to the **CellCtrlKey** property of a cell but the combo box with the specified key did not exist).
7. [Fixed] iGrid did not link a combo list to a cell by the combo key if you specified the key in the **CellCtrlKey** property before setting the cell type to **igCellCombo** or **igCellTextCombo** in the **CellType** property for the cell.

v6.50, build 0072 | 2017-Jun-16

1. [Enhancement] The copy-paste functionality has been improved. If several cells are selected in multi-select mode and you paste one cell value from the clipboard, this value is pasted into all selected cells.
2. [Enhancement] The **LayoutCol** property has been improved to support high-resolution screens. Now it saves the current system DPI in the layout string and process this value later when you restore the layout. If the current system DPI differs from the saved value when you restore the layout, the column widths saved in pixels are recalculated accordingly to provide the user with the columns of the same logical width.
3. [Enhancement] The tree sort algorithm and the **SortRowChildNodes** method have been improved. They work faster if your tree grid has sibling tree nodes already sorted. The performance increase can be up to 5 times – the more sorted sibling nodes you have, the better the effect. In addition to that, those sets of nodes are not reordered as it may have happened in the previous builds.
4. [Fixed] iGrid may have displayed string cell values improperly after assigning an empty string to the **CellFmtString** property (for instance, "(1,2)" was displayed as "-12").
5. [Fixed] The auto fit column width operation did not work properly if cell texts contained ampersand characters ("&"). The functionality of the **AutoWidthCol** and **AutoWidthCols** methods have been also fixed accordingly.
6. [Fixed] iGrid intercepted the F2, F4 and ENTER keys used to start editing even if the key wasn't applicable to the current cell (for instance, F4 can be used only to open the drop-down list in a combo box cell). As a result, these keys were not visible in the form's key processing events if the form's KeyPreview property was set to True.
7. [Fixed] The cell icon in text cells disappeared after editing the cell text.
8. [Fixed] The **SortRowChildNodes** method generated an error if the specified row did not have child rows.

v6.50, build 0063 (v6.5 release) | 2017-Mar-22

New and existing optimized members

1. [New] If you needed to set heights of all rows to the same new value, in the previous builds of iGrid you used code like this:

```
Dim iRow As Long

iGrid1.BeginUpdate

For iRow = 1 To iGrid1.RowCount
   iGrid1.RowHeight(iRow) = 30
Next

iGrid1.EndUpdate
```

This version of iGrid provides you with a much better way to do that in one call with the new **SetRowHeights** method. Now you can use the following statement instead of the code snippet above:

```
iGrid1.SetRowHeights 30
```

The new method allows you to not only shorten your code, but also increase its performance dramatically. The **SetRowHeights** method was optimized and can perform the same work ten thousands of times faster than setting the height of each row in a loop.

One of the classic cases when this method is extremely useful is when you provide your users with the ability to change the grid font. If you display only single-line texts in the cells, you can use the following code snippet to adjust the row heights after the user has selected a new font:

```
Set iGrid1.Font = objNewFont
iGrid1.SetRowHeights iGrid1.GetOptimalCellHeight()
```

The full signature of the **SetRowHeights** method is the following:

```
Sub SetRowHeights( _
   ByVal iHeight As Integer, _
   Optional ByVal vStartRow As Variant, _
   Optional ByVal vEndRow As Variant)
```

Its two optional parameters, **vStartRow** and **vEndRow**, can be used to limit the range of rows to apply the new height to. If **vStartRow** is omitted, the first row is implied; if **vEndRow** is omitted, the last row is used as the end row.

2. [New] Similar to the **SetRowHeights** method, the new method **SetColWidths** was implemented. It can be used to set the same width for every column in iGrid or in the specified range of columns in one call:

```
Sub SetColWidths( _
   ByVal lWidth As Long, _
   Optional ByVal vStartCol As Variant, _
   Optional ByVal vEndCol As Variant)
```

The optional **vStartCol** and **vEndCol** parameters allow you to specify the first and last column to process. Note that actually you specify the column indices using these parameters, but the user may reorder columns and columns with subsequent indicies may not be adjacent columns on the screen.

3. [New] iGrid implements the new **AutoHeightRows** method to automatically adjust the height of every row in iGrid or in the specified row range:

```
Sub AutoHeightRows( _
   Optional ByVal vStartRow As Variant, _
   Optional ByVal vEndRow As Variant, _
   Optional ByVal bIncludeInvisibleRows As Boolean = False, _
   Optional ByVal lMinimumHeight As Long = -1, _
   Optional ByVal lMaximumHeight As Long = -1, _
   Optional ByVal eCellVisibility As ECellVisibilityFilter = _
      igCellVisCurrentlyVisible)
```

The first two optional parameters, **vStartRow** and **vEndRow**, are used to specify the last and end row to process. If you omit these parameters, the first and last row are implied.

The **bIncludeInvisibleRows** parameter is used to tell the method that rows with the **RowVisible** property set to False must be processed. In most cases there is no need to automatically adjust height for invisible rows, and the method skips them unless you specify True in this parameter.

The remaining parameters are used the same way like in the **AutoHeightRow** method used to adjust the height of one row.

The new **AutoHeightRows** method is performed much faster compared to the equivalent construction with the **AutoHeightRow** method in a loop due to special optimizations. You get a big performance improvement on huge grids if you use the new method. The effect is similar to the use of the new **SetRowHeights** method instead of a loop with the **RowHeight** calls.

4. [New] Similar to the **AutoHeightRows** method, the new **AutoWidthCols** method was implemented. It allows you to adjust the width of every column or the column from the specified column range in one call:

```
Public Sub AutoWidthCols( _
   Optional ByVal vStartCol As Variant, _
   Optional ByVal vEndCol As Variant, _
   Optional ByVal bIncludeInvisibleCols As Boolean = False, _
   Optional ByVal lMinimumWidth As Long = -1, _
   Optional ByVal lMaximumWidth As Long = -1, _
   Optional ByVal eCellVisibility As ECellVisibilityFilter = _
      igCellVisCurrentlyVisible)
```

The optional **vStartCol** and **vEndCol** parameters allow you to specify the first and last column to process. By default invisible columns are not processed, but the **bIncludeInvisibleCols** parameter allows you to include them into the operation. The remaining parameters work the same way like in the **AutoWidthCol** method used to adjust the width of one column.

5. [New][Change][Optimization][Code-Upgrade] Starting from this version, by default iGrid no longer raises the following events that allow you to specify cell content dynamically: **CellDynamicText**, **CellDynamicIcons**, **CellDynamicFormatting** and **RowDynamicFormatting**. You need to enable these events explicitly with the new **DynamicContentEvents** property.

   This change in the iGrid functionality is related to the bad impact of these events on the control performance. The effect is especially noticeable in development environments like Microsoft Access because they use ActiveX controls through additional COM proxy layers. Turning off these events allowed us to enhance performance in some cases greatly. For instance, now the **AutoWidthCol** method or the equivalent operation initiated by a double-click on a column divider are performed 6 times faster in Microsoft Access. You may also notice other performance or visual improvements like less flickering due to the new policy for the dynamic content events listed above.

   The **DynamicContentEvents** property allows you to turn on the events individually for fine tuning of the grid performance. This property accepts values from the new **EDynamicContentEvents** enumeration:

```
Enum EDynamicContentEvents
   igDCEventNone = 0
   igDCEventCellDynamicFormatting = 1
   igDCEventCellDynamicIcons = 2
   igDCEventCellDynamicText = 4
   igDCEventRowDynamicFormatting = 8
End Enum
```

These values are used as flags to enable specific event individually. For instance, if you need to enable only the **CellDynamicText** event, use this setting:

```
iGrid1.DynamicContentEvents = igDCEventCellDynamicText
```

If you provide cell texts dynamically and apply color formatting to rows on-the-fly, use this setting:

```
iGrid1.DynamicContentEvents = igDCEventCellDynamicText + _
   igDCEventRowDynamicFormatting
```

You can enable/disable the dynamic content events from your code on-the-fly too. To disable all events, set the **DynamicContentEvents** property to its default value:

```
iGrid1.DynamicContentEvents = igDCEventNone
```

Note that only a small percent of real-world applications provides cell dynamic content, so in the vast majority of cases this change in the iGrid behavior will not affect your existing code.

6. [Optimization] The speed of tree sorting has been dramatically increased. The more rows you have, the more noticeable the effect. For instance, tree grids with 10'000-20'000 rows can be sorted 100 times faster now. The same optimization is in effect for the **SortRowChildNodes** method as well.

Changes related to combo lists

1. [New] The **FillFromRS** method for **ComboObject** was implemented. It allows you to quickly populate a combo list with the data from an ADO or DAO recordset. The method has the following signature:

   ```
   Public Sub FillFromRS( _
      ByVal RS As Object, _
      ByVal vTextField As Variant, _
      Optional ByVal vValueField As Variant)
   ```

   The **RS** parameter is used to pass the required recordset to the method. The **vTextField** parameter is used to specify the field from which combo item texts will be retrieved; it can be a string field name or its numeric index. The optional **vValueField** is used to specify the field from which combo item values will be retrieved. If **vValueField** is omitted, iGrid uses the field specified in the **vTextField** parameter to retrieve combo item values.

2. [New][Code-Upgrade] The **BeforeCommitEdit** event has a new parameter named **lComboListIndex** valid for cells with attached combo lists. This parameter has the Long data type and contains the zero-based index of the selected combo list item when the user tries to commit cell editing. For cells without combo lists or if an arbitrary text was entered into a combo box cell, this parameter contains -1.

3. [New] The **CellComboListIndex** property was introduced in this version of iGrid. It allows you to retrieve or set the zero-based index of the combo list item selected in a cell. The default value of this property is -1, which indicates that no item is selected in the cell. If you set this property to a value greater than the number of combo list items minus one or a negative value, the property will be reset to -1.

   For cells without attached combo lists, this property can be used to store extra cell values of the Long data type. The extra values are stored "as is" and are not reset to -1 like in the case of attached combo lists when you try to assign an invalid list index.

To sort iGrid by the **CellComboListIndex** values, the new sort type **igSortByComboListIndex** was added to the **ESortTypes** enumeration.

4. [New][Code-Upgrade] iGrid combo lists support a new mode in which their width is automatically set to the width of the column they are displayed in. This mode is turned on when the **ComboObject.Width** property is set to 0. This is also the new default value of this property instead of 120 pixels used in previous versions. If you set the combo list width explicitly in your code or call the **AutoAdjustWidth** method of the **ComboObject** to set the width implicitly, no changes in your source code are required when you upgrade to this version of iGrid.
5. [Enhancement] In the previous versions of iGrid the **ComboObject.AutoAdjustWidth** method always took into account the possible visibility of the vertical scroll bar in the list. As a result, you get a wider drop-down box if there was no vertical scroll bar. In this version the **AutoAdjustWidth** method always takes into account the visibility of the vertical scroll bar based on the current number of items and the value of the **MaxHeightInItems** property. Because of this, it is advised that you call the **AutoAdjustWidth** method after you populated the combo list and set its **MaxHeightInItems** property.

Other changes and enhancements

1. [New] The new **CellExtraData** property was implemented. It stores a user-defined extra value of the Long type for every cell. To sort iGrid by these values, use the new **igSortByExtraData** sort type.
2. [New] iGrid implements a new **BeforeRowFirstDraw** event raised when iGrid is about to draw a row for the first time. The index of the row is passed in the parameter of the event:

   ```
   Event BeforeRowFirstDraw(ByVal lRow As Long)
   ```

   This event is extremely useful if you implement a kind of virtual grid in which row data are requested once when they appear on the screen and then cached in the grid. To do this, set the corresponding cell values or other cell properties in an event handler of this event. iGrid does not redraw its contents after assignments to **CellValue** and other Cell* properties in this event, and you can set row data like in the code below without any side effects and additional **BeginUpdate**/**EndUpdate** calls:

   ```
   Private Sub iGrid1_BeforeRowFirstDraw(ByVal lRow As Long)
      iGrid1.CellValue(lRow, 1) = <data for column 1>
      iGrid1.CellValue(lRow, 2) = <data for column 2>
      iGrid1.CellValue(lRow, 3) = <data for column 3>
      ' Set cell properties for other columns
   End Sub
   ```

3. [New] This release of iGrid makes the earlier available helper **CTimer** class with some changes in member names an official public member. You can use this non-visual timer class to generate the **Timer** event at the specific intervals of time defined in its **Interval** property.
4. [New] This version of iGrid allows you to control the **VScrollBar.SmallChange** and **HScrollBar.SmallChange** values through the new **VScrollBar.SmallChangeDesiredValue** and **HScrollBar.SmallChangeDesiredValue** properties. By default, the **SmallChangeDesiredValue** property is set to 0 for each scroll bar, which means that iGrid uses the traditional automatic algorithms to set the **SmallChange** values depending on iGrid's properties. If you want to override the **SmallChange** value calculated automatically with a new value, assign it to the **SmallChangeDesiredValue** property.
5. [New] iGrid supports CTRL+A to select all cells in multi-select mode.
6. [Enhancement] If you populate iGrid from a recordset with the **FillFromRS** method, it automatically sets the **igSortByCheckState** sort type for Boolean columns displayed as check box cells to enable sorting by check box statuses on the column header clicks.

7. [Enhancement] The robustness of iGrid was improved. Now changing cell properties in events related to drawing (**CellDynamicText** and the like) does not lead to endless recursive calls of the drawing routine and eventually to the "Out of stack" error and possible IDE crash.
8. [Change][Code-Upgrade] The deprecated **bAllowSorting** parameter of the **Group** method was removed (this setting is controlled with the **AllowSorting** property in the latest builds of iGrid).
9. [Removed][Code-Upgrade] iGrid no longer supports old-fashioned 3D sort icons; the only available sort icons are solid triangles. As a result, the **ESortIconStyles** enumeration and the **Header.SortIconStyle** property were removed from iGrid.

Fixed bugs

1. [Fixed] The **BeginUpdate** method did not turn off redrawing in the header area. As a result, you may have seen changes in column headers regardless of cells during long operations like adjusting column widths and the like.
2. [Fixed] Some issues with the **TextEditSelStart** property that may led to improper text selection were resolved. Changing the **TextEditSelStart** property now automatically clears the selected text (the **TextEditSelLength** property is set to 0).
3. [Fixed] The CTRL+C combination did not work if iGrid was in read-only mode (the **Editable** property was set to False).
4. [Fixed] Changing the current cell in the **AfterContentsSorted** event may have caused problems with drawing in the cell area.
5. [Fixed] iGrid did not save the icon of the combo item in the cell if that item was found during text search while the user was entering text in the cells of the **igCellTextCombo** type.
6. [Fixed] Various problems may have occurred if an iGrid combo list contained more than 32'768 items.
7. [Fixed] The **FillFromRS** method did not detect DAO filed types properly. As a result, not all fields were displayed as expected. For instance, Boolean fields were displayed as True/False text cells instead of check box cells.
8. [Fixed] The **VScrollBar.SmallChange** property was not set to the **DefaultRowHeight** property on its change.
9. [Fixed] iGrid did not update the internal index of the combo list item corresponding to the cell value in column default cell objects after changes in combo lists.
10. [Fixed] The ampersand character was processed as the prefix character that underscores the next letter in combo list items. Now the combo list items are displayed "as is".
11. [Fixed] The height of combo box items was set to an enormous value on high-resolution screen.

v6.00, build 0032 | 2016-Aug-22

1. [Enhancement] This release of iGrid implements much better support for high DPI settings and Ultra-HD screens (4K monitors, etc.) Among these enhancements you will find:
   - All cell controls (check box, combo button, tree button) and sort icons in column headers are enlarged automatically if the application with iGrid uses high DPI. This does not apply to custom drawn tree buttons and outdated 3D sort icons drawn as a fixed pixel layout.
   - The values of the **TreeButtonSize** and **LevelIndent** properties are measured in device-independent pixels (DIPs) now. In iGrid, 1 DIP equals 1 physical pixel on a screen with 96 DPI. This allows iGrid to

enlarge the tree button and level indent automatically if the current DPI is higher than the standard value of 96. For instance, the default level indent value of 16 DIPs will be rendered as 40 pixels on a screen with 244 DPI when the system scale factor is 250%.

These high DPI enhancements are applied if the application is set as DPI-aware, which is true for all latest versions of the Microsoft Office VBA development environment. All compiled VB6 apps are not DPI-aware (unless they are provided with a special manifest file or the corresponding WinAPI call), but iGrid uses the new high DPI enhancements even in these applications if the user changes the OS DPI setting in old systems like Windows XP (so called Large Fonts, etc.) In fact, iGrid reads the current system DPI value available for the application and enlarges the items listed above if the DPI value is greater than 96.

2. [Enhancement] The default values for the **DefaultRowHeight** and **DefaultRowNormalCellHeight** properties are now calculated by iGrid automatically in its internal initialization code. These properties are set to the optimal height that is enough to display 1 line of cell text without clipping. When iGrid calculates the default value, it takes into account the current DPI setting available in the application, the grid font and other design-time settings for properties that may have influence on text output. This allows you to adjust row height in iGrid automatically and make it ready for high DPI settings with no coding in the most typical scenario, when only text data are displayed in it. Thus, now you can avoid writing the following line of code for every grid in your application if you display only text data in it:

   ```
   iGrid1.DefaultRowHeight = iGrid1.GetOptimalCellHeight()
   ```

   The **DefaultRowHeight** and **DefaultRowNormalCellHeight** properties are longer available for design-time usage and were hidden from the property browser because of this enhancement. If they remained available in the property browser, any design-time changes would be overridden by the optimal default values calculated as described above. But of course, you can set the **DefaultRowHeight** and **DefaultRowNormalCellHeight** properties to the required value explicitly in your code like in the statement above. This assignment will override the optimal row height calculated when iGrid is initialized because your code is executed after that.

   Note for Microsoft Office VBA developers. If you have already used iGrid ActiveX in VBA projects, you may still see the **DefaultRowHeight** and **DefaultRowNormalCellHeight** properties in the property editor. This happens because the VBA environment caches the public interfaces of used ActiveX controls in exd files. To solve this issue, just remove all copies of the iGrid600_10Tec.exd file on your system drive.

3. [Fixed] The text rows of design-time info (control name and its version) were drawn overlapped on Ultra-HD monitors.

4. [Fixed] iGrid used a lot of CPU resources in multiselect mode when the user changed the current cell, and this was especially noticeable in a remote desktop environment.

v6.00, build 0024 | 2016-Apr-27

1. [Fixed] If a cell of the type **igCellTextCombo** contained the Null value, iGrid crashed when the user clicked the combo button in the cell.

2. [Fixed] The grouping algorithm did not ignore hidden rows (group rows were created for values from hidden rows, aggregate functions were calculated for hidden rows, etc.)

3. [Fixed] If iGrid had the visible horizontal scroll bar with the scroll box in the last possible position, a try to shrink a column with the mouse launched fast automatic shrinking to the zero width.

4. [Fixed] The **SelItems** object did not report the selected items properly after sorting in row mode.

v6.00, build 0020 | 2016-Feb-17

1. [Enhancement] The speed of the **AddCol** method has been increased. Now it works much faster when adding several tens of columns.
2. [Enhancement] The **vRowBefore**, **vRowParent** parameters of the **AddRow** method and the **vColBefore** parameter of the **AddCol** method can be set to the Variant Empty value to tell iGrid that the corresponding parameter is not specified. This can simplify your code as now you can write just one **AddRow** or **AddCol** method call if one of the parameters listed above can be omitted depending on some conditions.
3. [Enhancement] A special triple buffering optimization technique has been implemented in the drawing code. This enhancement eliminates the tearing effect while scrolling the grid in the horizontal direction, when some first rows may have been drawn in the new position while the other rows remained on their places.

   Due to this enhancement, the drawing code works faster while scrolling the grid contents in both directions and iGrid requires 25% less CPU resources at that.
4. [Enhancement] Cells and rows are selected faster in multiselect mode. The effect of animation, when rows become selected row-by-row, was also removed.
5. [Fixed] The iGrid header may have scrolled faster than the cell area during horizontal scrolling.
6. [Fixed] Accessing the **KeySearchCol** and **RowTextStartCol** properties in non-VB/VBA development environments like Rockwell FactoryTalkView may have raised errors.
7. [Fixed] The contents of iGrid were scrolled in the horizontal and/or vertical direction during the drag select operation even if the grid did not have the corresponding scroll bar.
8. [Fixed] The **LoadFromArray** method did not populate combo cells properly.
9. [Fixed] iGrid did not recognize alpha-numeric keys after selecting items in combo lists with the mouse.
10. [Fixed] iGrid crashed if an array was stored as the value of a combo box cell.
11. [Fixed] The **Combos.Clear** and **Combos.Remove** methods failed with the run-time error ‘438’: Object doesn’t support this property or method.
12. [Fixed] iGrid had some issues with cell selection if columns had been reordered.
13. [Fixed] iGrid did not draw tree lines properly after hiding rows with the **RowVisible** property.
14. [Fixed] The VB IDE crashed if code execution stopped on a breakpoint in an **AfterCommitEdit** event handler for combo box cells.
15. [Fixed] An application with iGrid crashed if a message box was displayed from a **BeforeCommitEdit** event handler for a combo box cell.
16. [Fixed] Some minor issues related to virtual mode were fixed (the vertical scroll bar properties may have been set improperly in some cases, etc.)