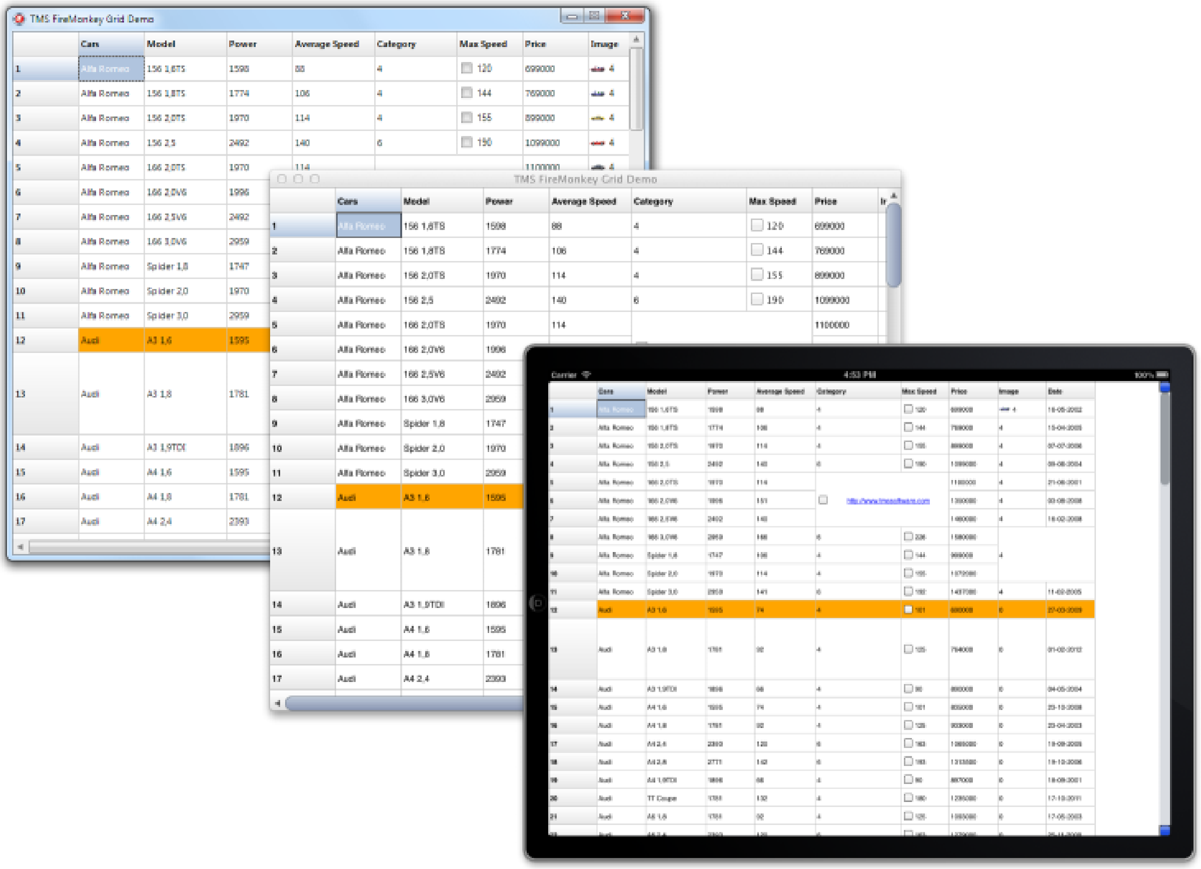
TMS Grid for FireMonkey DEVELOPERS GUIDE

February 2015

Index

Introduction

The TMS Grid for FireMonkey offers a fully cross-platform, high-performing, versatile and feature packed grid for the Embarcadero cross-platform framework FireMonkey. As such, it supports Win32, Win64, Mac OSX and iOS application development. It is from the ground up designed to respect the philosophy of style-able controls. At the same time, it is sufficiently similar to the VCL TAdvStringGrid to make developers used to TAdvStringGrid quickly familiar and up & running.



	Cars	Model	Power	Average Speed	Category	Max Speed	Price	Image
1	Alfa Romeo	156 1,6TS	1598	88	4	☐ 120	699000	4
2	Alfa Romeo	156 1,8TS	1774	106	4	☐ 144	769000	4
3	Alfa Romeo	156 2,0TS	1970	114	4	☐ 155	899000	4
4	Alfa Romeo	156 2,5	2492	140	6	☐ 190	1099000	4
5	Alfa Romeo	166 2,0TS	1970	114			1100000	4
6	Alfa Romeo	166 2,0V6	1996					
7	Alfa Romeo	166 2,5V6	2492					
8	Alfa Romeo	166 3,0V6	2959					
9	Alfa Romeo	Spider 1,8	1747					
10	Alfa Romeo	Spider 2,0	1970					
11	Alfa Romeo	Spider 3,0	2959					
12	Audi	A3 1,6	1595					
13	Audi	A3 1,8	1781					
14	Audi	A3 1,9TDI	1896					
15	Audi	A4 1,6	1595					
16	Audi	A4 1,8	1781					
17	Audi	A4 2,4	2393					

IMPORTANT NOTICE:
If the FireMonkey framework is new to you, please see the chapter “General FireMonkey component usage guidelines” that offers an introduction that is recommended to read before you start working with the TMS Grid for FireMonkey. Another interesting source of information is http://docwiki.embarcadero.com/RADStudio/en/FireMonkey_Application_Platform

Grid properties

ColumnCount: integer: Gets or sets the number of columns displayed in the grid.

Columns: TTMSFMXGridColumns: A collection of columns to allow designtime / runtime customization and persistence of grid cell layout / types and behavior such as sorting and editing. More information about the columns collection can be found in the “Columns” chapter.

DefaultColumnWidth: single: The default width of a column. When new columns are added to the grid, the width will default to this value. The width of a column can be changed per column with grid.ColumnWidths[ACol]: single.

DefaultRowHeight: single: The default height of a row. When new rows are added to the grid, the height will default to this value. The height of a row be changed per row with grid.RowHeights[ARow]: single.

FixedColumns: integer: Gets or sets the amount of fixed columns in the grid. Fixed columns are columns that remain visible at all times, that do not scroll along with the grid when scrolling horizontal and that get a separate appearance, the fixed cell appearance.

FixedFooterRows: integer: Gets or sets the amount of fixed footer rows. Footer rows are rows that are positioned at the bottom side of the grid, remain visible at all times, do not scroll along with the grid when scrolling vertical and that get a separate appearance, the fixed cell appearance.

FixedRightColumns: integer: Gets or sets the amount of fixed right columns. Right columns are columns which are positioned at the right side of the grid, remain visible at all times and do not scroll along with the grid when scrolling horizontal.

FixedRows: integer: Gets or sets the amount of fixed rows in the grid. Fixed rows are rows which remain visible at all times and do not scroll along with the grid when scrolling vertical.

LeftCol: integer: Gets or sets the index of the first visible normal column that is selectable. Use this property to programmatically control the horizontal scroll position in the grid.

Options: The various options available in the grid. (Explained in “Options” chapter)

RowCount: integer: Gets or sets the amount of rows in the grid.

ScrollMode: TTMSFMXGridScrollMode: Gets or sets the type of scrolling. There are 2 types of scrolling: cellscrolling and pixelscrolling. With cellscrolling is selected, the scrolling is based on entire columns or rows. A complete row or column is moved depending on the scroll direction. With pixelscrolling is selected, the scrolling is based on the total width and height of the cells allowing you to scroll more precisely and having cells partially visible.

SelectionMode: TTMSFMXGridSelectionMode: Gets or sets the type of selection with mouse or keyboard that is allowed in the grid. The selection varies from single to multiple cells, column and row selections, disjunct selections.

smNone: Hides selection, all other interaction remains active
smSingleCell: Selects a single cell. When changing selection, the previous cell state returns to normal.
smSingleRow: Selects a complete row. When changing selection, the previous row state returns to normal.
smSingleColumn: Selects a complete column. When changing selection, the previous column state returns to normal.
smCellRange: Enables selecting multiple cells. When performing a shift-click, the range between the previous cell and current cell is selected. A range of cells can also be selected when holding and dragging the mouse over the grid.
smRowRange: Enables selecting multiple rows. When performing a shift-click, the range between the previous row and current row is selected. A range of rows can also be selected when holding and dragging the mouse over the grid.
smColumnRange: Enables selecting multiple columns. When performing a shift-click, the range between the previous column and current column is selected. A range of columns can also be selected when holding and dragging the mouse over the grid.
smDisjunctRow: Has the same functionality as smRowRange, and with the ability to distinct select rows with the ctrl key.
smDisjunctColumn: Has the same functionality as smColumnRange and with the ability to distinct select columns with the ctrl key.
smDisjunctCell: Has the same functionality as smCellRange and with the ability to distinct select cells with the ctrl key.

TopRow: integer: Gets or sets the first normal visible row that is selectable. This property can be used to programmatically control the vertical scroll position in the grid.

UseColumns: Boolean: public property used to toggle between persisted column data through the columns collection at designtime/runtime (UseColumns = true) or dynamically created data at runtime (UseColumns = false). More information about the Columns collection can be found in the “Columns” chapter.

Options

The options persistent class property hierarchically exposes many of the grid's settings for different areas of usage:

Bands

BandRowCount: integer: The amount of alternative colored rows (bands). The appearance of
the band row is controlled by the style.
Enabled: Boolean: Enables banding on the grid. When not enabled, all rows have the normal appearance.
NormalRowCount: integer: The amount of normal colored rows.

Borders

CellBorders: TTMSFMXGridBorders: Sets which borders (vertical, horizontal or all) are visible on normal cells.
FixedCellBorders: TTMSFMXGridBorders: Sets which borders (vertical, horizontal or all) are visible on fixed cells.

Clipboard

AllowColGrow: Boolean: Automatically adds the amount of columns if necessary when pasting data of more cells than can fit into the grid.
AllowRowGrow: Boolean: Automatically adds the amount of rows if necessary when pasting data of more cells than can fit into the grid.
Enabled: Boolean: Enables copy & paste shortcuts (Ctrl-X, Ctrl-V, Ctrl-C) at runtime
IgnoreReadOnly: Boolean: Enables or disables a paste or cut operation for readonly cells.
PasteAction: TTMSFMXGridClipboardPasteAction: 2 options: overwrites the data of the cells within range of the clipboard data, or inserts new rows and columns from the focused cell

Columns

Editing

AutoComplete: Boolean: Enables autocompletion in the edit field when the editortype is set to use an edit or an edit with button (etEdit, etEditBtn inplace editor types) and adds a possibility to display the autocomplete list with a popup or directly in the edit area.
AutoCompleteItems: TStringList: The items that appear when autocompletion is enabled.
AutoHistory: Boolean: Automatically adds the text when editing is finished to the AutoCompleteItems list.
Enabled: Boolean: Enables or disables editing in the grid.

AutoPost: Livebindings only, automatically posts the edited data to the database when the dataset is in edit mode.
AutoCancel: Livebindings only, automatically cancels the dataset when in edit mode.

Filtering

DropDown: Boolean: Shows a dropdown button on the fixed row that is set with
DropdownFixedRow: integer: Selects the fixed row where the filter dropdown appears in case there are more than one fixed row. By default, the filter dropdown appears in the first fixed row (0)
DropDownHeight: integer: Gets or sets the height of the filter dropdown list.
DropDownWidth: integer: Gets or sets the width of the filter dropdown list.
MultiColumn: boolean: allows automatic multicolumn filtering.
Rows:TTMSFMXGridFilterRows: Filtering applies to all cells in a specific filtered column or only the normal cells, which exclude summary, fixed and node cells. By default, only normal row cell values are used in the filter operation.

Footer

Visible: Boolean: When true, shows the footer area of the grid.

Grouping

AutoCheckGroup: Boolean: When true and rows, including the group header rows have checkboxes, a click on the group header row's checkbox, will check/uncheck all rows within the group.
AutoSelectGroup:Boolean: When true, a click on the group header row will perform a selection of all rows within the group. For this automatic selection of rows within a group to work, the SelectionMode should be either smCellRange or smDisjunctRow
GroupCalcFormat: string: sets the format to use to display a group calculation result in the group summary row.
MergeHeader: Boolean: when true, the header row of a group is automatically merged.
MergeSummary: Boolean: when true, the summary row of a group is automatically merged.
ShowGroupCount: Boolean: when true, the number of rows within a group is automatically displayed in the group header.
Summary: Boolean: when true, when grouping is applied, a summary row is automatically added for each group.

Header

Visible: Boolean: When true, shows the header area of the grid.

IO

AlwaysQuotes: Boolean: When true, all text is exported within double quotes when exporting to CSV files. When false, only cell text that contains a space character or the delimiter character is surrounded by double quotes.
Delimiter: Char: Gets or sets the delimiter character used to export/import CSV files. When Delimiter equals #0, the grid will try to determine the used column delimiter itself and will use the default ',' delimiter to export, otherwise it will use the specified Delimiter.

QuoteEmptyCells: Boolean: When true, an empty cell is exported as “”, otherwise it is exported as just empty text.
XMLEncoding: string: Gets or sets the XML encoding attribute for the generated XML file during export. By default, XMLEncoding is: ISO-8859-1
SaveVirtualCellData: Saves data set in OnGetCellData to format of choice when exporting.

HTMLExport

BorderSize: integer: sets the HTML border size of the HTML table exported from the grid.
CellPadding: integer: sets the HTML cell padding used in the HTML table exported from the grid.
CellSpacing: integer: sets the HTML cell spacing used in the HTML table exported from the grid.
ConvertSpecialChars: Boolean: when true, special characters like &, “, … will be exported as HTML special characters &amp; , &quot; etc...
ExportImages: Boolean: when true, images used in the grid will be exported as separate set of images in the subfolder \Images where the HTML file is generated.
FooterFile: string: specifies a HTML file that will be included as footer in the generated HTML file.
HeaderFile: string: specifies a HTML file that will be included as header in the generated HTML file.
NonBreakingText: Boolean: exports spaces in cell text as &nbsp; to ensure there is no automatic text breaking in the exported HTML.
PrefixTag: string: prefix that is rendered just before the HTML table.
SaveColors: Boolean: when true, all colors are exported to HTML.
SaveFonts: Boolean: when true, all font settings per cell are exported to HTML.
Show: Boolean: when true, the generated HTML file is automatically shown with the default viewer on the operating system after generation with grid.SaveToHTML().
SuffixTag: string: suffix that is rendered after the HTML table.
Summary: string: sets the HTML summary tag attribute text.
TableStyle: string: sets the HTML table attributes
Width: Boolean: sets the width as % of the generated HTML table.
XHTML: Boolean: when true, generates HTML table according to XHTML spec.

Keyboard

AllowCellMergeShortCut: Boolean: When true, this enables the shortcuts Ctrl-M and Ctrl-S to perform merging & splitting of selected cells.
ArrowKeyDirectEdit: Boolean: Enables or disables direct editing when navigating with the arrowkeys left / right / up / down in the editor. When the up / down key is pressed the previous / next row cell is selected in the same column. When left / right key is pressed the previous / next column cell is selected in the same row. When pressing the left / right key the selection is only changed when the cursor is at the end of the text or the beginning of the text.
DeleteKeyHandling: TTMSFMXGridDeleteKeyHandling Enables or disables deleting a row in the grid.

EnterKeyDirectEdit: Boolean: Enables or disables direct editing when pressing the enter key in the previous cell and when EnterKeyHandling property is set.
EnterKeyHandling: TTMSFMXGridEnterKeyHandling:Sets the way of handling the enter key after editing. After pressing the enter key you can move to the next column or row, and optionally choose if the columncount / rowcount needs to be increased when pressing enter at the end of the column / row.
InsertKeyHandling: TTMSFMXGridInsertKeyHandling: Enables or disables inserting a row in the grid.
PageScrollSize: integer: Sets the amount of cells that are scrolled when pressing pagedown or pageup. The PageScrollSize property is 0 by default which means that the PageScrollSize is automatically calculated based on the size of the grid and the size of the cells.
TabKeyDirectEdit: Boolean: Enables or disables direct editing when pressing the tab key in the previous cell.
TabKeyDirection: TTMSFMXGridTabKeyDirection:Sets the direction when pressing the tab key on a cell. The next cell will be located the next column or the next row.
TabKeyHandling: TTMSFMXGridTabKeyHandling:Sets what the tab key handling should do when at the end of a row / column or when at the beginning or end of the grid. Here the tab key can move to the next control in the application, remain inside the grid or use a mixed mode where the next control is focused if the tab key is pressed on the last cell / first cell of the grid depending on the Direction and if the shift key is pressed.

Lookup

CaseSensitive: Boolean: Enables or disables case sensitive lookup.
Enabled: Boolean: Enables or disables lookup. When lookup is enabled, editing must be disabled in order to function properly.
Incremental: Boolean:Adds the typed character to an internally used lookup string that is used to search after the correct cell inside the focused column. When Incremental is false, the internal lookup string is set empty each time a key is pressed. The lookup will then only search for text in cells starting with the last typed character.

Mouse

AutoDragging: Boolean: Enables or disables auto dragging when performing column dragging / row dragging. When enabled the grid performs automatic dragging, in the direction where the mouse is going, when the mouse is outside the grid. The further the mouse is removed from the edges the faster the scrolling will occur.
AutoScrolling: Boolean: Enables or disables auto scrolling. When enabled the grid performs automatic scrolling, in the direction where the mouse is going, when the mouse is outside the grid. The further the mouse is removed from the edges the faster the scrolling will occur.
AutoScrollingInterval: integer: Sets the interval of the timer that takes care of the automatic scrolling.
AutoScrollingSpeed: integer: Sets the speed of the automatic scrolling.
ColumnDragging: Boolean: Enables column dragging. When pressing and dragging a column, the cursor changes and a column can be swapped with a different column.

ColumnSizing: Boolean: Enables column sizing. When hovering with the mouse over the normal grid cells. The cursor will change if the column can be sized. This can also be controlled with events.
DirectComboDrop: Boolean: Enables direct combo drop when clicking on a cell and the editor is set to a combo type editor
DirectEdit: Boolean: Enables direct editing when clicking on a cell. When DirectEdit is false.The cell must be selected first and then clicked again to allow editing.
FixedColumnSizing: Boolean: Enables fixed column sizing. When hovering with the mouse over the fixed grid cells. The cursor will change if the column can be sized. This can also be controlled with events.
FixedRowSizing: Boolean: Enables fixed row sizing. When hovering with the mouse over the fixed grid cells. The cursor will change if the row can be sized. This can also be controlled with events.
RowDragging: Boolean:Enables row dragging. When pressing and dragging a row, the cursor changes and a row can be swapped with a different row.
RowSizing: Boolean: Enables row sizing. When hovering with the mouse over the fixed grid cells. The cursor will change if the row can be sized. This can also be controlled with events.
TouchScrolling: Boolean: Enables or disables touch scrolling. With touch scrolling, the area of the grid can be used to scroll. This can only be used in combination with single cell, row or column selectionmode.
TouchScrollingSensitivity: single: The sensitivity of the touch scrolling.
WheelScrollSize: integer: The amount of cells that are scrolled when using the mouse-wheel to navigate.

Printing

DescriptionColor: Color of the font used when printing a description on a page.
DescriptionFont: The font used when printing a description on a page.
Description: string: The description used on a page.
DescriptionPosition: TTMSFMXGridPrintPosition: The position of the description.
PageNumberColor: Color of the font used when printing a pagenumber on a page.
PageNumberFont: The font used when printing a pagenumber on a page.
PageNumberFormat: string: The format of the page number.
PageNumberPosition: TTMSFMXGridPrintPosition: The position of the page number.
PrintCellBackGround: Boolean: Sets if the cell background must be painted or not and whether only the fixed and normal cell background needs to be drawn or the cell background as is.
PrinterOrientation: TPrinterOrientation: Sets the page orientation of the printer.
PrintMargins: TBounds: Sets the margins used for drawing the grid. The grid automatically calculates the amount of pages necessary to print the complete grid or a selection of it.
RepeatFixedColumns: Boolean: Repeats the fixed columns on each page.
RepeatFixedRows: Boolean: Repeats the fixed rows on each page.
TitleColor: Color of the font used when printing a title on a page.
TitleFont: The font used when printing a title on a page.
Title: string: The title used on a page.
TitlePosition: TTMSFMXGridPrintPosition: the position of the title.

Rendering

Mode: the mode used to render the cells in the grid. By default all cells that contain children, controls are rendered

Sorting

BlankPosition: TTMSFMXGridSortBlankPosition: determines where empty (blank) cells should be positioned during sorting, ie. blFirst as first cells in ascending sorts or blLast as last cells in ascending sorts.
Columns: TTMSFMXGridSortColumns: determines what columns will be sorted. By default, scAll is set, meaning that when sorting is performed, the cells in all columns will be reordered by the sort. When set to scNormal, fixed column cells will not be affected by the sort. When set to scSingle, only the cells of the column for which the sort is performed will be reordered.
FixedColumns: Boolean: when true, fixed column header cells can be clicked as well to trigger a sort.
IgnoreBlanks: Boolean: when true, sorting will ignore spaces inside the text for comparisation
IgnoreCase: Boolean: When true, perform sorting always without case sensitivity.
Mode: TTMSFMXGridSortingMode: selects whether sorting by clicking on fixed column header cells is possible or not and whether it triggers a single column or multi column sort.
MultiColumn: boolean: When true, multiple columns can be defined as sort criteria. The primary sort column is set with a simple click, additional secondary sort columns are set with shift click.

URL

Color: TAlphaColor: sets the color hyperlinks in the grid.
Full: Boolean: when true, the hyperlink is displayed with its protocol identifier. When false, the protocol identifier is hidden, the URL without protocol identifier is shown in URL color and optionally underlined in the grid cell.
Open: Boolean: when true, automatically open the hyperlink when click in the default browser.
Show: Boolean: when true, automatically show hyperlinks (i.e. text with prefix http://, file://, ftp://, nntp://, mailto: as hyperlinks.
Underline: Boolean: when true, automatically show hyperlinks in cells underlined

Scrolling

VerticalScrollBarVisible: Boolean: Shows or hides the vertical scrollbar.
HorizontalScrollBarVisible: Boolean: Shows or hides the horizontal scrollbar.

Organisation

In its basic layout, a grid is a matrix of cells with mainly fixed cells (not editable) and normal cells. A fixed cell will not scroll along with normal cells and thus remain visible on any of the 4 sides of the grid. This number of fixed rows and/or columns on the 4 sides of the grid is controlled by properties: grid.FixedRows, grid.FixedColumns, grid.FixedFooterRows, grid.FixedRightColumns. In addition to fixed, non scrolling rows and/or columns, the grid can also perform column freezing. These are columns or rows of normals cells that will not scroll along with the other columns or rows in the grid. The number of freeze columns and rows is set with grid.FreezeColumns, grid.FreezeRows. Cells are accessible via grid.Cells[Column,Row]:string and the selected cell(s) can be set with properties:

```
grid.Selection := CellRange(StartCol,StartRow,EndCol,EndRow);
grid.FocusedCell := Cell(Col,Row);
```

The grid features several selection modes: single cell selection, single row selection, single column selection, cell range selection, row range selection, column range selection, disjunct row selection, disjunct cell selection and disjunct column selection. The selection mode is chosen with the property:

```
grid.SelectionMode: TTMSFMXGridSelectionMode;
```

The scroll position in the grid can be programmatically set or retrieved via the properties grid.LeftCol: integer, grid.TopRow: integer.
Note that scrolling in the grid can be performed in two ways: cell scrolling and pixel level scrolling. In cell scrolling mode, the minimum quantity of a scroll is an entire column or row, in pixel scrolling mode, scrolling is per pixel and can thus be done on sub cell level. The scrolling mode is controlled by the property:

```
grid.ScrollMode = (smCellScrolling, smPixelScrolling)
```

When navigating through the grid, the grid will automatically scroll when selecting a cell that is partially visible and bring it in view. When clicking and dragging the mouse outside of the grid normal cells area, the grid will start an autoscroll operation which will scroll with a delta that is automatically calculated based on the distance of the mouse to the last position inside the grid.

This automatic scrolling and some additional properties to control speed and interval can be set under grid.Options.Mouse.

The size of columns & rows is controlled by grid.ColumnWidths[ColumnIndex]: single, grid.RowHeights[RowIndex]: single and it can be configured that the user can resize columns or rows at runtime with: grid.Options.Mouse.ColumnSizing, grid.Options.Mouse.FixedColumnSizing, grid.Options.Mouse.RowSizing, grid.Options.Mouse.FixedRowSizing.

The amount of displayed columns and rows are set with grid.ColumnCount: integer and grid.RowCount: integer properties.

Rows and columns can be inserted / deleted by pressing the Insert / Delete key on the keyboard. Note that when a row is inserted or deleted from the user interface, the events OnCanInsertRow, OnInsertRow, OnCanDeleteRow,OnDeleteRow are triggered. The OnCanInsertRow, OnCanDeleteRow events occur before the actual insert or delete happens and have the extra parameter Allow: Boolean that can be set to false to disallow a specific row insert or delete.

Programatically, following methods are available inserting, deleting columns or rows but also to move and swap columns or rows:

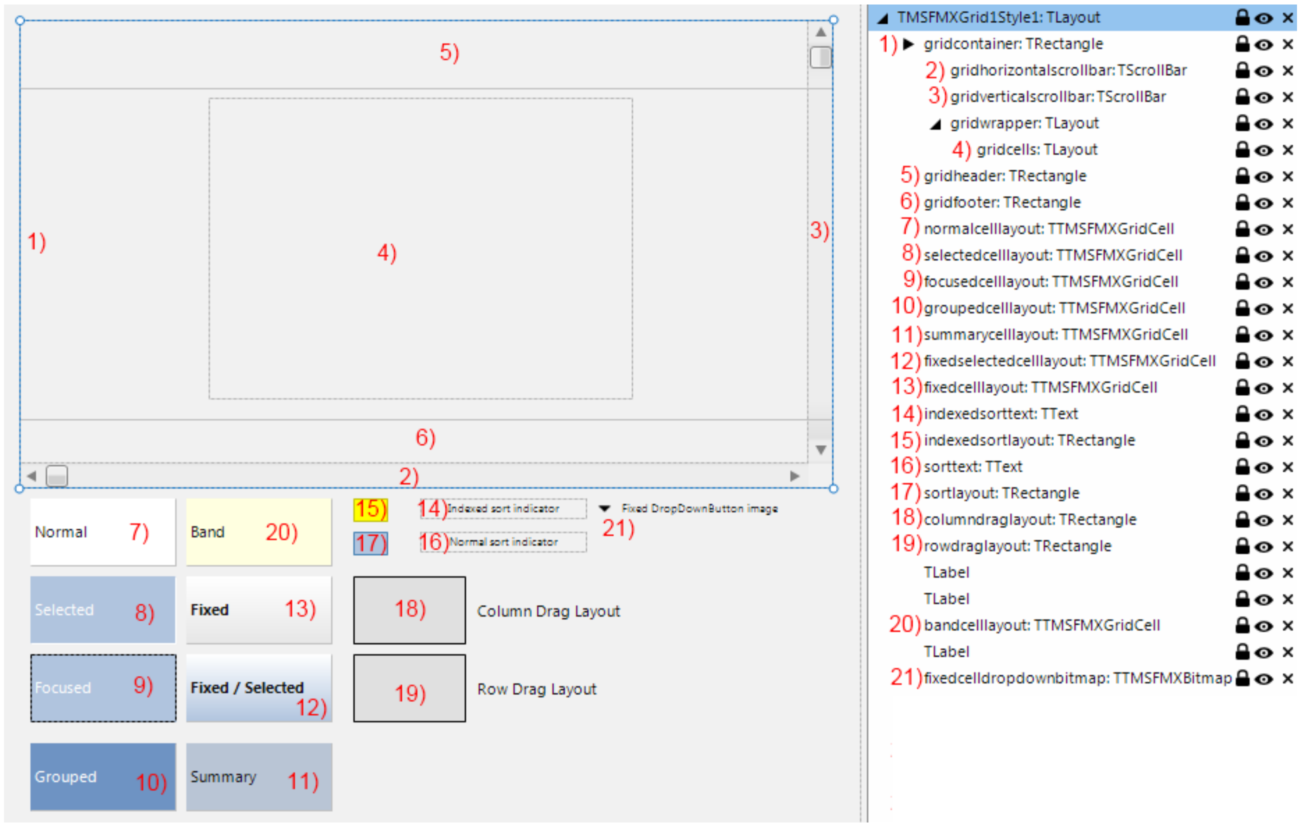
grid.InsertRow(ARow: integer): insert a new row at row ARow
grid.InsertRows(ARow, NumRows: integer): insert NumRows rows at row ARow
grid.DeleteRow(ARow: integer) : remove row with index ARow from the grid
grid.DeleteRows(ARow, NumRows: integer): remove NumRows rows at row ARow.
grid.InsertColumn(ACol: integer): insert a new column at column ACol
grid.DeleteColumn(ACol: integer): remove column with index ACol from the grid
grid.MoveRow(FromRow, ToRow: integer): move row from index FromRow to index ToRow
grid.MoveColumn(FromCol, ToCol: integer): move column from index FromCol to index ToCol
grid.SwapRows(Row1,Row2: integer): swap content of Row1 and Row2
grid.SwapColumns(Col1,Col2: integer): swap content of Col1 and Col2

The keyboard interaction can be modified with the grid.Options.Keyboard.InsertKeyHandling and grid.Options.Keyboard.DeleteKeyHandling properties.

Columns and Rows can be moved to another position by clicking on the fixed column / row and dragging them to a different position. When dragging, a visual copy is made of the column and is then moved transparently on the grid. When releasing the column or row is swapped with the column or row on the new position. The Column and row dragging can be enabled in the grid.Options.Mouse.ColumnDragging and grid.Options.Mouse.RowDragging properties.

Styling

The TMS Grid for FireMonkey offers a completely style-able look and feel. When dropping the component on the form you will notice various elements that are all separately style-able. To apply a different style, right-click the component and click “Edit Custom Style” or “Edit Default Style” depending if you want to apply the new style to all new grid instances or only to the current selected grid. More on styling can be found in the “FireMonkey Styles” chapter.

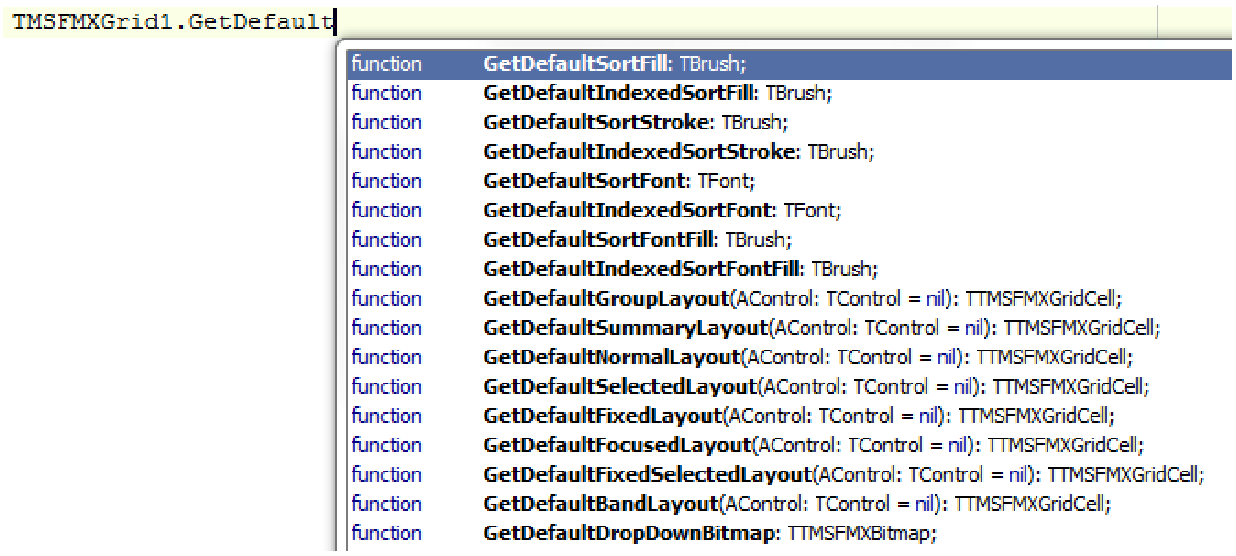


```
TMSFMXGrid1Style1: TLayout
1) gridcontainer: TRectangle
    2) gridhorizontalscrollbar: TScrollBar
    3) gridverticalscrollbar: TScrollBar
    gridwrapper: TLayout
        4) gridcells: TLayout
5) gridheader: TRectangle
6) gridfooter: TRectangle
7) normalcelllayout: TTMSFMXGridCell
8) selectedcelllayout: TTMSFMXGridCell
9) focusedcelllayout: TTMSFMXGridCell
10) groupedcelllayout: TTMSFMXGridCell
11) summarycelllayout: TTMSFMXGridCell
12) fixedselectedcelllayout: TTMSFMXGridCell
13) fixedcelllayout: TTMSFMXGridCell
14) indexedsorttext: TText
15) indexedsortlayout: TRectangle
16) sorttext: TText
17) sortlayout: TRectangle
18) columndraglayout: TRectangle
19) rowdraglayout: TRectangle
    TLabel
    TLabel
20) bandcelllayout: TTMSFMXGridCell
    TLabel
21) fixedcelldropdownbitmap: TTMSFMXBitmap
```

1) The container of the grid that contains all elements.
2) The horizontal scrollbar of the grid.
3) The vertical scrollbar of the grid.
4) The container of the cells.
5) The header of the grid.
6) The footer of the grid.
7) The normal cell layout.
8) The layout of a cell that is selected.
9) The layout of a cell that is selected and focused.
10) The layout for a header row when using grouping.
11) The layout for a summary row when using grouping.
12) The layout for a fixed cell that indicates where a normal cell is selected.

13) The layout for a fixed cell.
14) The indexed sort indicator text displayed when using indexed sorting.
15) The indexed sort indicator layout.
16) The normal sort indicator text displayed when using normal sorting.
17) The normal sort indicator layout.
18) The drag layout when performing a column drag operation.
19) The drag layout when performing a row drag operation.
20) The layout for the alternate rows when using banding.
21) The bitmap used in the dropdown when using filtering and the dropdown button is visible on a fixed cell.

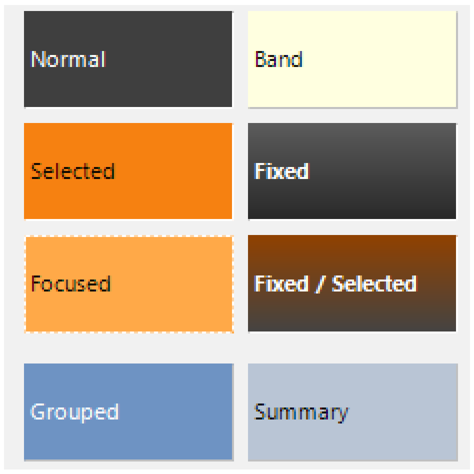
Each element can also be accessed programmatically with a function grid.GetDefault*. If an item is not accessible by the grid directly, you can access it by using grid.FindStyleResource('style name of the element').

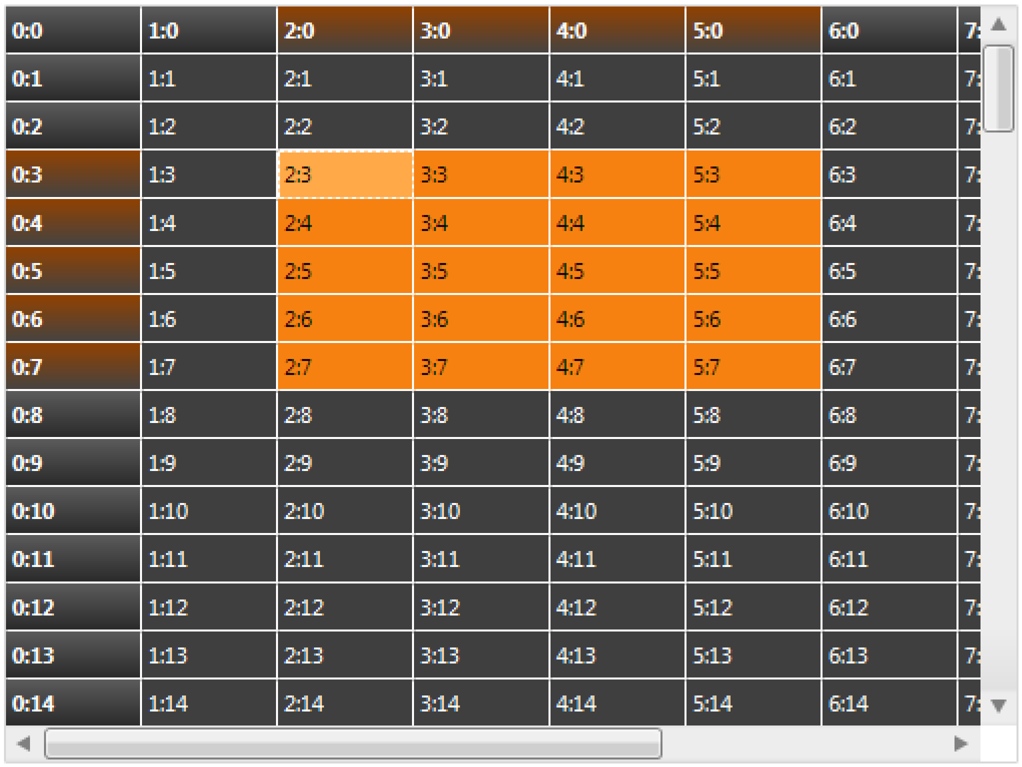


```
TMSFMXGrid1.GetDefault
function    GetDefaultSortFill: TBrush;
function    GetDefaultIndexedSortFill: TBrush;
function    GetDefaultSortStroke: TBrush;
function    GetDefaultIndexedSortStroke: TBrush;
function    GetDefaultSortFont: TFont;
function    GetDefaultIndexedSortFont: TFont;
function    GetDefaultSortFontFill: TBrush;
function    GetDefaultIndexedSortFontFill: TBrush;
function    GetDefaultGroupLayout(AControl: TControl = nil): TTMSFMXGridCell;
function    GetDefaultSummaryLayout(AControl: TControl = nil): TTMSFMXGridCell;
function    GetDefaultNormalLayout(AControl: TControl = nil): TTMSFMXGridCell;
function    GetDefaultSelectedLayout(AControl: TControl = nil): TTMSFMXGridCell;
function    GetDefaultFixedLayout(AControl: TControl = nil): TTMSFMXGridCell;
function    GetDefaultFocusedLayout(AControl: TControl = nil): TTMSFMXGridCell;
function    GetDefaultFixedSelectedLayout(AControl: TControl = nil): TTMSFMXGridCell;
function    GetDefaultBandLayout(AControl: TControl = nil): TTMSFMXGridCell;
function    GetDefaultDropDownBitmap: TTMSFMXBitmap;
```

Note: Modifications can be done programmatically in the form's OnCreate of the application only after the grid.UpdateStyle is called. This is necessary to force the FireMonkey framework to load the correct style template for the grid first. When the app is running, the grid.UpdateStyle call is not necessary since the grid has already loaded the style.

Below is a sample of a grid that is styled, showing the cell layout and the result:





0:0	1:0	2:0	3:0	4:0	5:0	6:0	7:
0:1	1:1	2:1	3:1	4:1	5:1	6:1	7:
0:2	1:2	2:2	3:2	4:2	5:2	6:2	7:
0:3	1:3	2:3	3:3	4:3	5:3	6:3	7:
0:4	1:4	2:4	3:4	4:4	5:4	6:4	7:
0:5	1:5	2:5	3:5	4:5	5:5	6:5	7:
0:6	1:6	2:6	3:6	4:6	5:6	6:6	7:
0:7	1:7	2:7	3:7	4:7	5:7	6:7	7:
0:8	1:8	2:8	3:8	4:8	5:8	6:8	7:
0:9	1:9	2:9	3:9	4:9	5:9	6:9	7:
0:10	1:10	2:10	3:10	4:10	5:10	6:10	7:
0:11	1:11	2:11	3:11	4:11	5:11	6:11	7:
0:12	1:12	2:12	3:12	4:12	5:12	6:12	7:
0:13	1:13	2:13	3:13	4:13	5:13	6:13	7:
0:14	1:14	2:14	3:14	4:14	5:14	6:14	7:

Cell Properties

Cells[Col,Row: Integer]: string

Sets the text of a cell. Col & row are the visual column & row coordinates of the cell.

Hello World!

Floats[Col,Row: Integer]: double

Gets or sets the value of a cell as a double.

AllCells[Col,Row: Integer]: string

Similar behavior as Cells[Col, Row: Integer]: TCellData
This accesses the cells with column & row coordinates irrespective of column or row hiding. Coordinates are real column, row coordinates.

StrippedCells[Col,Row: Integer]: string

Returns the text of a cell with all HTML formatting removed.

AllFloats[Col,Row: Integer]: double

Gets or sets the value of a cell as a double. This accesses the cells with column & row coordinates irrespective of column or row hiding where grid.Floats[Col,Row] access the value based on displayed cell coordinates.

45.4

ColumnWidths[Col: Integer]: single

Gets or sets the width of a column. When no column width is set for a column, the width is set to DefaultColumnWidth.

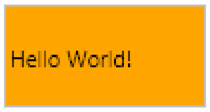
Hello World!

RowHeights[Row: Integer]: single

Gets or sets the height of a row. When no row height is set for a row, the height is set to DefaultRowHeight

Colors[Col,Row: Integer]: TAlphaColor

Sets a background color on the cell.

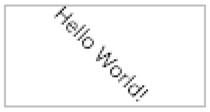


Note that it is also possible to set the cell color dynamically via the event OnGetCellLayout.

```
procedure TForm4.TMSFMXGrid1GetCellLayout(Sender: TObject; ACol, ARow:
Integer;
  ALayout: TTMSFMXGridCellLayout; ACellState: TCellState);
begin
  if (ACol = 3) and (ARow >= TMSFMXGrid1.FixedRows) then
    ALayout.Fill.Color := claRed;
end;
```

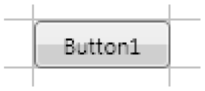
Angles[Col,Row: Integer]: single

Sets an angle of the text in a cell in degrees.



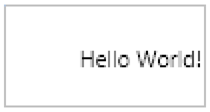
CellControls[Col,Row: Integer]: TFMXObject

Adds any FireMonkey control to a cell.



HorzAlignments[Col,Row: Integer]: TTextAlign

Changes the horizontal text alignment in a cell.

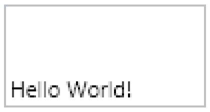


Note that is also possible to dynamically set the cell text alignment via the event OnGetCellLayout:

```
procedure TForm4.TMSFMXGrid1GetCellLayout(Sender: TObject; ACol, ARow:
Integer;
  ALayout: TTMSFMXGridCellLayout; ACellState: TCellState);
begin
  if ARow < TMSFMXGrid1.FixedRows then
    ALayout.TextAlign := TTextAlign.taCenter;
end;
```

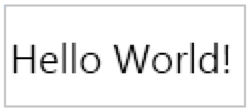
VertAlignments[Col,Row: Integer]: TTextAlign

Changes the vertical text alignment in a cell.



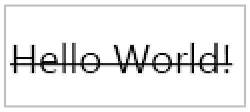
FontSizes[Col,Row: Integer]: single

Changes the size of the font of the text in a cell.



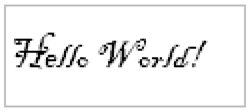
FontStyles[Col,Row: Integer]: TFontStyles

Changes the style of the font of the text in a cell.



FontNames[Col,Row: Integer]: string

Changes the name of the font of the text in a cell.



FontColors[Col,Row: Integer]: TAlphaColor

Changes the color of the font of the text in a cell.

Note that the cell font can also be set dynamically with the event OnGetCellLayout. In this sample, the font is set red for negative values in the grid:

```
procedure TForm1.TMSFMXGrid1GetCellLayout(Sender: TObject; ACol, ARow:
Integer;
  ALayout: TTMSFMXGridCellLayout; ACellState: TCellState);
var
  cv: Double;
begin
  if (ACol >= TMSFMXGrid1.FixedColumns) and (ARow >= TMSFMXGrid1.FixedRows)
then
  begin
    cv := TMSFMXGrid1.AllFloats[ACol,ARow];
    if cv < 0 then
      ALayout.FontFill.Color := claRed
    else
      ALayout.FontFill.Color := claGreen;
  end;
end;
```

ReadOnlys[Col,Row: Integer]: boolean

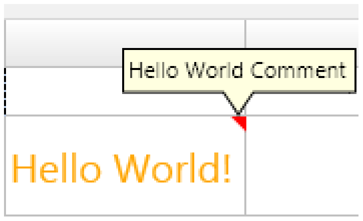
Sets a cell readonly, which means that the cell is no longer editable via the user-interface. The equivalent event for dynamically controlling this is OnGetCellReadOnly.

Objects[Col,Row: Integer]: TObject

Adds a reference to an object that can be used to link to a cell. Note that the application is responsible for the lifetime of the object and should as such destroy the object when needed. All the grid provides is a reference to any TObject instance.

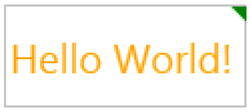
Comments[Col,Row: Integer]: string

Adds a comment cell to the grid, when clicking on the comment triangle, a popup is shown with the comment that has been added to a cell. By default the comment is indicated by a red triangle in the top right corner of the cell.



CommentColors[Col,Row: Integer]: TAlphaColor

Sets the color of the comment triangle.



Events

OnCustomCompare:Event called when format type is set to ssCustom through the OnSortFormat: Event triggered during sorting to allow to dynamically specify what sort method the grid should use for a specific column. By default, the grid tries to automatically determine the type of data in cells and can as such detect regular text, numbers and dates.

OnIOProgress:Event called when exporting or importing data to indicate the progress of the operation.

OnRawCompare: Event called when format type is set to ssRaw through the OnSortFormat event. This allows defining at application level the compare method to use for a sort operation.

OnCellsChanged: Event called when the values of a cell change for example as result of a clipboard cut / paste action, a file import etc…

OnClipboardBeforePasteCell: Event called before pasting the data in the cell. The event passes a Value parameter that can be modified per cell as well as an Allow parameter to allow the text to be inserted in the cell.

OnClipboardAfterPasteCell: Event called after pasting the data in the cell. Here the value is passed after the data has been pasted from the clipboard.

OnClipboardPaste: Event called when pasting the data in the range of selected cells. Through this event the selection can be retrieved.

OnNeedFilterDropDown: Event called when applying filtering, to allow / disallow filtering on a column. By default the dropdown is shown by setting grid.options.filtering.dropdown := True. Through this event, the dropdown can additionally be hidden for specific columns.

OnNeedFilterDropDownData: Event called to fetch the data displayed in the filter list. When the dropdown is active and the filtering can be applied (retrieved through an optional combination of grid.options.filtering.dropdown and the OnNeedFilterDropDown event), the data can be modified, items can be added to and removed from the filtering display list.

OnFilterSelect:Event called when clicking on an item in the filter list.

OnLoadCell:Event called when loading data into the grid. Allows to dynamically modify a cell value after it was retrieved from a file, for example to decrypt data.

OnSaveCell:Event called when saving data from the grid. Allows to dynamically modify the value that will be stored in a file during a save, for example to encrypt data.

OnSelectCell: Event called before selecting a cell, to specify if a cell can be selected or not with an var Allow: Boolean parameter.

OnGetCellClass: Event called to return the type of cell that is used inside the grid. Allows to customize the type of any cell in the grid. There are some predefined cell types that can be used and that implement controls such as a checkbox, radio button and bitmap.

The sample code below adds a grid cell with a bitmap. With the OnGetCellProperties the cell can be casted to the cellclasstype defined in the OnGetCellClass event to change properties and to add a bitmap.

```
procedure TForm738.TMSFMXGridLiveBinding1GetCellClass(Sender: TObject;
ACol,
  ARow: Integer; var CellClassType: TFmxObjectClass);
begin
  if (ACol = 4) and (ARow = 3) then
    CellClassType := TTMSFMXBitmapGridCell;
end;
```

OnGetCellControl: Event called to get the control that is added inside the cell. This event differs from the OnGetCellClass event in such way that it is an additional control that can be added and doesn't replace the original cell. The default cell type is TTMSFMXGridCell. Through this event, a control reference that already exists (is placed on the form) can be displayed in the cell. Note that all controls are client-aligned inside the cell.

OnGetCellData: Event called when loading the data that is displayed inside the cell. Allows to dynamically change the text displayed in a cell or to implement virtual cells.

OnGetCellProperties: Event called to apply additional properties dynamically to the cell. Through this event, properties can be applied that are unique per column, row or cell. This event is called simultaneously with the OnGetCellAppearance, but to keep a clean overview it is recommended to apply all non-cell visual related properties through this event.

OnGetCellAppearance: Event called to apply additional appearance settings dynamically to the cell. The Cell object parameter can be casted to the TTMSFMXGridCell type or depending on the type of cell added, casted to a different implementation type of TTMSFMXGridCell such as TTMSFMXCheckGridCell or TTMSFMXBitmapGridCell. With this event, the layout can be modified of the cell per state that can be retrieved through the ACellState parameter.

```
procedure TForm1.TMSFMXGrid1GetCellAppearance(Sender: TObject;
  ACol, ARow: Integer; Cell: TFmxObject; ACellState: TCellState);
begin
  case ACellState of
    csNormal: (Cell as TTMSFMXGridCell).Layout.Fill.Color := claRed;
    csFocused: (Cell as TTMSFMXGridCell).Layout.Fill.Color := claBlue;
    csFixed: (Cell as TTMSFMXGridCell).Layout.Fill.Color := claGreen;
    csFixedSelected: (Cell as TTMSFMXGridCell).Layout.Fill.Color :=
claOrange;
    csSelected: (Cell as TTMSFMXGridCell).Layout.Fill.Color :=
claLime;
```

```
  end;
end;
```

OnGetCellLayout: A more simple version of the event OnGetCellAppearance, called to apply additional appearance settings to the cell. Here the ALayout parameter is a direct reference to (Cell as TTMSFMXGridCell).Layout property.

OnGetCellMergeInfo: Event called to get the merge information of a cell. Used to display merged cells.

OnGetCellReadOnly: Event called to set a cell read-only. With the AReadOnly parameter a cell can be set readonly, this means that the cell data cannot be changed by editing, or by pasting data inside the cell.

OnGetRowIsBand: Event called to set if a row is alternate and needs to use the banding layout. The banding layout can be found in the stylebook when editing the custom or default style.

OnCanInsertRow: Event called if a row can be inserted in the grid or not.

```
procedure TForm1.TMSFMXGrid1CanInsertColumn(Sender: TObject;
  ACol: Integer; var Allow: Boolean);
begin
  Allow := ACol = 5; //allows inserting when the active row is 4 and
the inserted row will be inserted on column index 5
end;
```

OnCanAppendRow: Event called if a row can be appended (added) to the grid or not.

```
procedure TForm1.TMSFMXGrid1CanAppendRow(Sender: TObject;
  ARow: Integer; var Allow: Boolean);
begin
  Allow := ACol <= 11; //allows appending one column
end;
```

OnCanAppendColumn: Event called if a column be appended (added) to the grid or not.

Example with 10 columns:

```
procedure TForm1.TMSFMXGrid1CanAppendColumn(Sender: TObject;
  ACol: Integer; var Allow: Boolean);
begin
  Allow := ACol <= 11; //allows appending one column
end;
```

OnCanDeleteRow: Event called if a row can be deleted in the grid.

Example with 10 columns:

```
procedure TForm1.TMSFMXGrid1CanDeleteRow(Sender: TObject;
  ACol: Integer; var Allow: Boolean);
begin
  Allow := ARow > 4; //does not allow deleting the first 5 rows
end;
```

OnInsertRow: Event called after a row is inserted.

OnAppendRow: Event called after a row is appended.

OnAppendColumn: Event called after a column is appended.

OnDeleteRow: Event called after a row is deleted.

OnCellAnchorClick: Event called if a cell with an anchor is clicked.

OnGetCellEditorCustomClassType: Event called to specify a custom class type for an inplace editor that is not directly supported by the grid.

Sample with column index 4 and row index 2 returns a custom editor of TTreeview type.

```
procedure TForm1.TMSFMXGrid1GetCellEditorType(Sender: TObject;
  ACol, ARow: Integer; var CellEditorType: TTMSFMXGridEditorType);
begin
  if (ACol = 4) and (ARow = 2) then
    CellEditorType := etCustom;
end;

procedure TForm1.TMSFMXGrid1GetCellEditorCustomClassType(
  Sender: TObject; ACol, ARow: Integer;
  var CellEditorCustomClassType: TFmxObjectClass);
begin
  if (ACol = 4) and (ARow = 2) then
    CellEditorCustomClassType := TTreeView;
end;
```

OnGetCellEditorType: Event called to specify the type of an inplace editor. If the type is set to etCustom, the OnGetCellEditorCustomClassType event is called.

OnCellEditGetData: Event called to predefine the value set in the inplace editor.

Cellstring that is set in the inplace editor is ‘Hello World’.

```
procedure TForm1.TMSFMXGrid1CellEditGetData(Sender: TObject; ACol,
  ARow: Integer; CellEditor: TFmxObject; var CellString: string);
```

```
begin
  CellString := 'Hello World';
end;
```

OnCellEditValidateData: Event called to validate the value coming from the inplace editor. After the OnCellEditGetData is called and the cellstring is set in the edit, this event is called when the editing stops. The value that comes from the editor and is ready to be inserted in the cell can be validated and modified through this event.

OnCellEditSetData: Event called to set the data in the cell. Through this event when validation returns true, the value can be modified one last time before the data is inserted in the cell.

OnCellEditGetColor: Event called to predefine the color set in the inplace editor. Similar to the OnCellEditGetData, this event passes a color parameter that can be changed before the color is passed to the editor. This event is only used when setting the correct editor type in the OnGetCellEditorType. The etColorPicker and etColorComboBox are editor types that will trigger this event instead of the Data variant.

OnCellEditValidateColor: Event called to validate the color coming from the inplace editor. In the same way as the data variant, the selected color can be validate and modified.

OnCellEditSetColor: Event called to set the color in the cell. When validation is true, the value is used as a background color for the cell. The color of the cell can be retrieved with grid.Colors[ACol, ARow: Integer]: TAlphaColor;

OnCellEditDone: Event called when editing is finished.

OnGetCellEditorProperties: Event called before the inplace editor is shown to apply additional properties. Depending on the chosen editor type in the OnGetCellEditorType event, the CellEditor parameter must be casted to the correct editor class type. Below is a list for the supported editor types:
et*Edit: all: TTMSFMXEdit class type.
et*EditBtn: TTMSFMXEditBtn class type
etComboBox: TComboBox class type
etComboEdit: TComboEdit class type
etSpinBox: TSpinBox class type.
etDatePicker: TCalendarBox class type.
etDateEdit: TCalendarEdit class type.
etColorPicker: TComboColorBox class type.
etColorComboBox: TColorComboBox class type.
etTrackBar: TTrackBar class type.
etArcDial: TArcDial class type.
etCustom: class type passed through OnGetCellEditorCustomClassType

OnGetCellIsFixed: Event called to return if a normal cell is fixed or not. When this event returns true for a normal cell, the cell cannot be selected or modified and has the fixed cell layout applied.

OnFixedCellBitmapClick: Event called when clicking on a bitmap of a fixed cell. The return type for the OnGetCellClass is TTMSFMXBitmapGridCell.

OnFixedCellButtonClick: Event called when clicking on a button of a fixed cell. The return type for the OnGetCellClass is TTMSFMXFixedGridCell and the showbutton property set through OnGetCellProperties is true.

OnFixedCellDropDownButtonClick: Event called when clicking on the dropdownbutton of a fixed cell. The return type for the OnGetCellClass is TTMSFMXFixedGridCell and the showdropdownbutton property set through OnGetCellProperties is true and the OnNeedFilterDropDown event allows showing the dropdown button on a fixed cell.

OnFixedCellCheckBoxClick: Event called when clicking on the checkbox of a fixed cell. The return type for the OnGetCellClass is TTMSFMXFixedGridCell and the showcheckbox property set through OnGetCellProperties is true or a checkbox has been added to the header with grid.AddHeaderCheckBox(Col, Row: Integer; State: Boolean);

OnFixedCellSpinBoxChange: Event called when the value of the spinbox has changed of a fixed cell. The return type for the OnGetCellClass is TTMSFMXFixedGridCell and the showspinbox property set through OnGetCellProperties is true.

OnCellBeforeDraw: Event called before the cell is drawn. Through this event custom drawing can be done, before the actual content of the cell is drawn, for example to draw a different background or to add additional painting under the background and text. With the AllowDraw parameter set to false the complete cell is not drawn, with the ADrawBackGround parameter set to false the background is not drawn and the same applies to the ADrawText parameter. The ARect is the full cell rectangle and the ATextRect parameter is the rectangle for the text, taking optionally enabled cell controls in to calculation.

```
procedure TForm1.TMSFMXGrid1CellBeforeDraw(Sender: TObject; ACol, ARow:
Integer;
  ACanvas: TCanvas; var ARect, ATextRect: TRectF; var ADrawText,
  ADrawBackGround, AllowDraw: Boolean);
begin
  AllowDraw := False;
end;
```

OnCellAfterDraw: Event called after the cell is drawn. Event that can be used in the same way as the OnCellBeforeDraw event, but after all cell content is drawn, here there are no Allow parameters because the cell is already been painted. Through this event, additional painting can be done above all the already painting cell content.

OnCellBitmapClick: Event called when clicking on the bitmap of a cell. The return type for the OnGetCellClass is TTMSFMXBitmapGridCell or the cell has been added with grid.AddBitmap or grid.AddBitmapName procedures.

OnCellButtonClick: Event called when clicking on the button of a cell. The return type for the OnGetCellClass is TTMSFMXButtonGridCell.

OnCellRadioButtonClick: Event called when clicking on the radiobutton of a cell. The return type for the OnGetCellClass is TTMSFMXRadioGridCell or the cell has been added with grid.AddRadioButton or grid.AddRadioButtonColumn procedures.

OnCellShowPopup: Event called when showing the popup of a cell. The return type for the OnGetCellClass is TTMSFMXFixedGridCell or TTMSFMXCommentGridCell and the cell has filtering enabled in case of TTMSFMXFixedGridCell or has set a comment in case of TTMSFMXCommentGridCell and the popup is shown by clicking the dropdown button or the comment triangle.

OnCellHidePopup: Event called when hiding the popup of a cell.

OnCellCheckBoxClick: Event called when clicking on the checkbox of a cell. The return type for the OnGetCellClass is TTMSFMXCheckGridCell or the cell has been added with grid.AddCheckBox, grid.AddCheckBoxColumn or grid.AddHeaderCheckBox procedures.

OnCellCommentClick: Event called when clicking on the comment triangle of a cell. The return type for the OnGetCellClass is TTMSFMXCommentGridCell or the cell has been added with grid.Comments[ACol, ARow: Integer].

OnCellSortClick: Event called when the fixed sort column is clicked. The return type for the OnGetCellClass is TTMSFMXFixedGridCell, sorting is enabled and the fixed cell has been clicked to apply sorting.

OnCellNodeClick: Event called when clicking on the node of a cell. The return type for the OnGetCellClass is TTMSFMXNodeGridCell or the cell has been added with grid.AddNode.

OnCanSizeColumn: Event called when a column is about to be sized. Set Allow parameter to false if sizing of a specific column needs to be blocked.

OnCanSizeRow: Event called when a row is about to be sized. Set Allow parameter to false if sizing of a specific row needs to be blocked.

OnColumnSize: Event called while the column is being sized. The NewWidth parameter can be modified when sizing to limit the size of the column.

OnRowSize: Event called while the row is being sized. The NewHeight parameter can be modified when sizing to limit the size of the row.

OnColumnSized: Event called when the column is done sizing.

OnRowSized: Event called when the row is done sizing.

OnColumnSorted: Event called when the column is sorted.

OnCanSortColumn: Event called when a column is about to be sorted. Set the Allow parameter to false if sorting of a specific column should be blocked.

OnCellClick: Event called when a cell is clicked.

OnFixedCellClick: Event called when a fixed cell is clicked.

OnPrintBegin: Event called when printing begins.

OnPrintNewPage: Event called when printing starts a new page.

OnPrintDrawCell: Event called when printing a cell on the page.

OnPrintBeforeDrawCell: Event called before printing a cell on the page. This event has identical parameters as the OnCellBeforeDraw but is separated to avoid interference with the grid when printing.

OnPrintAfterDrawCell: Event called after printing a cell on the page. This event has identical parameters as the OnCellAfterDraw but is separated to avoid interference with the grid when printing.

OnPrintProgress: Event called during printing with the current progress in percentage.

OnPrintEnd: Event called when the printing has ended.

OnPrintEndPage: Event called when the printing has ended a page.

OnPrintDrawTitle: Event called when drawing the title. The Allow parameter can be used to disable drawing a title, as well as the AText parameter to modify the text that is drawn.

OnPrintDrawDescription: Event called when drawing the description. The Allow parameter can be used to disable drawing a title, as well as the AText parameter to modify the text that is drawn.

OnPrintDrawPageNumber: Event called when drawing the page number. The Allow parameter can be used to disable drawing a title, as well as the AText parameter to modify the text that is drawn.

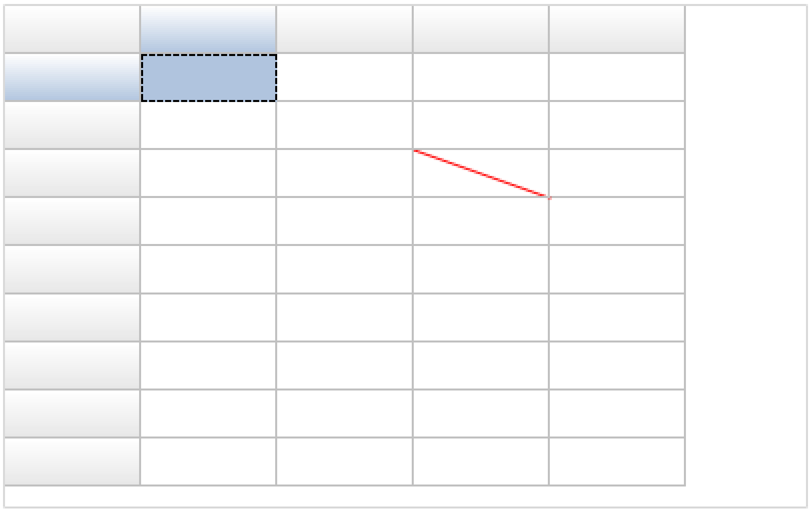
OnPrintBeforeDraw: Event called before the printing starts drawing.

OnPrintAfterDraw: Event called after the printing has done drawing.

Custom Cell Drawing

Each cell supports custom drawing via an event and with event parameters to optionally disable the background, text, as well as a reference to the Canvas and the rectangle. With the events OnCellBeforeDraw and OnCellAfterDraw you can custom draw on a cell, or complete column / row of choice that can be retrieved by using the Row and Column parameters. Below is a sample that draws a diagonal line as a background which replaces the default background of a cell on location 3, 3.

```
procedure TForm1.TMSFMXGrid1CellBeforeDraw(Sender: TObject; ACol,
  ARow: Integer; ACanvas: TCanvas; var ARect, ATextRect: TRectF; var
ADrawText,
  ADrawBackGround, AllowDraw: Boolean);
begin
  if (ACol = 3) and (ARow = 3) then
  begin
    ADrawBackGround := False;
    ACanvas.Stroke.Color := claRed;
    ACanvas.DrawLine(ARect.TopLeft, ARect.BottomRight, 1);
  end;
end;
```



Custom Cell Class

When dropping a default grid on the form, you will notice fixed and normal cells. Based on the design philosophy of firemonkey, we have decided that each cell is a separate control instead of painting all cells on a canvas. The basic implementation supports a fill, stroke and a text. For a normal default cell, the cell class type is TTMSFMXGridCell. A fixed cell implements and inherits all features from the base cell and adds the possibility to add controls that will help you in terms of filtering, checking a complete column or additional functionality that you can provide with the various events that are implemented. The fixed cell class is TTMSFMXFixedGridCell and is used when the grid detects a cell is fixed. The default grid cell can be changed to a different type through events or with the correct procedures in the grid. The grid cell already supports a number of different classes that are listed below.

The cell class type can be set dynamically via the event OnGetCellClass or various methods are provided to set this programmatically. Using the OnGetCellClass, actually any type of FireMonkey class can be used as cell class. The grid already offers a number of predefined cell classes for the most common uses and events like OnGetCellLayout can deal with these built-in classes to properly set cell properties as color, font, alignment. When using a class type not known to the grid, it will be required to dynamically control properties such as color, font, etc… via the OnGetCellProperties and cast the parameter Cell: TFmxObject to the type specified for the cell.

Example: specifies that a checkbox is used for column 3:

```
procedure TForm4.TMSFMXGrid1GetCellClass(Sender: TObject; ACol, ARow:
Integer;
  var CellClassType: TFmxObjectClass);
begin
  if (ARow >= TMSFMXGrid1.FixedRows) and (ACol = 3) then
    CellClassType := TTMSFMXCheckGridCell;
end;
```

The equivalent code to programmatically add checkboxes is:

```
var
  i: integer;
begin
  for i := TMSFMXGrid1.FixedRows to TMSFMXGrid1.RowCount - 1 do
    TMSFMXGrid1.AddCheckBox(3,i,false);
end;
```

To dynamically change a property of such checkbox cells when needed, the code that could be used is:

```
procedure TForm1.TMSFMXGrid1GetCellProperties(Sender: TObject; ACol,
  ARow: Integer; Cell: TFmxObject);
begin
```

```
  if (ACol = 3) and (ARow >= TMSFMXGrid1.FixedRows) and (Cell is
TTMSFMXCheckGridCell) then
  begin
    (Cell as TTMSFMXCheckGridCell).CheckBox.IsChecked := true;
  end;
end;
```

TTMSFMXGridCell

Basic implementation of a grid cell, with a fill, stroke and text properties.

TTMSFMXRadioGridCell

Inherits from TTMSFMXGridCell and adds the capability to display a radiobutton.

The methods to add & remove radiobuttons are:

TMSFMXGrid.AddRadioButton(Col,Row,Index: integer; State: boolean = false);
Adds a radiobutton in cell Col,Row belonging to group Index. The State parameter sets the default state of the radiobutton.

TMSFMXGrid.AddRadioButtonColumn(Col,Index: integer);
Adds a column of radiobuttons as group Index

TMSFMXGrid.RemoveRadioButton(Col,Row: integer);
Removes the radiobutton from cell Col,Row

TMSFMXGrid.IsRadioButton(Col,Row: integer): boolean;
Returns true when the cell contains a radiobutton

TMSFMXGrid.RadioButtonState(Col,Row: integer): boolean;
Returns the state of a radiobutton in cell Col,Row

Index parameter: The index of a radio group to which the radiobutton belongs
State parameter: Sets the radiobutton in a checked or unchecked state.

Examples:

```
TMSFMXGrid1.AddRadioButton(1, 1, 4, True);
TMSFMXGrid1.AddRadioButtonColumn(1, 1);
```

TTMSFMXCheckGridCell

Inherits from TTMSFMXGridCell and adds the capability of displaying a checkbox.

The methods to add & remove checkboxes are:

TMSFMXGrid.AddHeaderCheckBox(Col,Row: integer; State: boolean = false);
Adds a checkbox in a fixed column header cell. A column header checkbox will toggle the checkbox state of all checkboxes in a column when it is clicked.

TMSFMXGrid.AddCheckBox(Col,Row: integer; State: boolean = false);
Add a checkbox to cell Col,Row.

TMSFMXGrid.AddCheckBoxColumn(Col: integer);
Add checkboxes in all cells of column Col.

TMSFMXGrid.RemoveCheckBox(Col,Row: integer);
Remove the checkbox in cell Col,Row.

TMSFMXGrid.IsCheckBox(Col,Row: integer): boolean;
Returns true when the cell Col,Row contains a checkbox.

TMSFMXGrid.CheckBoxState[Col,Row: integer]: boolean
Gets or sets the checkbox state of cell Col,Row

TMSFMXGrid.AddDataCheckBox(Col,Row: integer; State: boolean = false);
Adds a data checkbox to cell Col,Row. A data checkbox cell is a cell with a checkbox where the checked state of the checkbox reflects the text value of the cell. When the text value of the cell equals TMSFMXGrid.CheckTrue, it will be displayed as checked. When the text value of the cell equals TMSFMXGrid.CheckFalse, it will be displayed as unchecked. To get or set the checkbox state of this checkbox type, use:

```
TMSFMXGrid.Cells[Col,Row] := TMSFMXGrid.CheckTrue.
```

or

```
if TMSFMXGrid.Cells[Col,Row] = TMSFMXGrid.CheckTrue then
  // checkbox is true
```

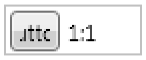
TMSFMXGrid.AddDataCheckBoxColumn(Col: integer);
Adds data checkboxes in all cells of column Col.

Examples:

```
TMSFMXGrid1.AddCheckBox(1, 1, True);
TMSFMXGrid1.AddCheckBoxColumn(1);
```

TTMSFMXButtonGridCell

Inherits from TTMSFMXGridCell and adds the capability of displaying a button.



The methods to add & remove buttons in the grid are:

TMSFMXGrid.AddButton(Col,Row: integer; AText: string; AWidth: integer = 24);
Adds a button to the grid at cell Col,Row with caption text AText and width AWidth.

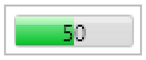
TMSFMXGrid.IsButton(Col,Row: integer): boolean;
Returns true when cell Col,Row contains a button

TMSFMXGrid.RemoveButton(Col,Row: integer);
Removes the button from cell Col,Row

When clicked, the button in the cell triggers the event OnCellButtonClick.

TTMSFMXProgressGridCell

Inherits from TTMSFMXGridCell and adds the capability of displaying a progressbar.



Progressbar values are between 0 and 100.

The methods to add & remove progress bars in the grid are:

TMSFMXGrid.AddProgressBar(Col,Row: Integer; Value: Single);
Adds a progress bar with position Value in the grid cell Col,Row.

TMSFMXGrid.AddDataProgressBar(Col,Row: Integer);
Adds a data progress bar in the grid cell Col,Row. The value of the progressbar is controlled by the value set in grid.Cells[Col,Row].

TMSFMXGrid.SetProgressBarValue(Col,Row: Integer; Value: single);
Sets the value of a progressbar in cell Col,Row.

TMSFMXGrid.GetProgressBarValue(Col,Row: integer): single;
Retrieves the value of a progressbar in cell Col,Row.

TMSFMXGrid.IsProgressBar(Col,Row: Integer): boolean;
Returns true when cell Col,Row contains a progress bar.

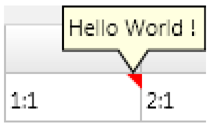
TMSFMXGrid.RemoveProgressBar(Col,Row: Integer);
Removes the progressbar from cell Col,Row.

Examples:

```
TMSFMXGrid1.AddProgressBar(1, 1, 50);
```

TTMSFMXCommentGridCell

Inherits from TTMSFMXGridCell and adds the capability of display a comment in a popup. Also adds an indicator in the topright corner.



The comment text and comment indicator triangle color can also be controlled by properties:

TMSFMXGrid.Comments[Col,Row]: string;
TMSFMXGrid.CommentColors[Col,Row]: TAlphaColor;

When the comment text is an empty string, no comment triangle will be displayed.

Examples:

```
TMSFMXGrid1.Comments[1, 1] := ‘Hello World!’;
TMSFMXGrid1.CommentColors[1, 1] := claRed;
TMSFMXGrid1.Comments[2, 2] := ‘’; // remove comment from cell 2,2
```

TTMSFMXFixedGridCell

Inherits from TTMSFMXGridCell and adds several capabilities such as showing a sorting indicator, a filter dropdown button, a column header checkbox.

TTMSFMXRotatedTextGridCell

Inherits from TTMSFMXFixedGridCell and adds the capability to rotate the text.

The angle of rotation in a rotated text cell is controlled by the property:

TMSFMXGrid.Angles[Col,Row]: single;

TTMSFMXNodeGridCell

Inherits from TTMSFMXGridCell and adds the capability of displaying a node with which several rows can be collapsed or expanded.

The methods to deal with nodes in the grid are:

TMSFMXGrid.AddNode(Row, Span: Integer);
Adds a node that spans Span rows in cell 0,Row.

TMSFMXGrid.RemoveNode(Row: Integer);
Removes a node from cell 0,Row.

TMSFMXGrid.IsNode(Row: Integer): boolean;
Returns true when cell 0,Row contains a node.

TMSFMXGrid.SetNodeState(Row: Integer; State: TNodeState);
Sets the state of the node in cell 0,Row as opened or closed.
TNodeState = (nsClosed, nsOpen);

TMSFMXGrid.GetNodeState(Row: integer): TNodeState;
Returns the state of a node in cell 0,Row with TNodeState = (nsClosed, nsOpen);

TMSFMXGrid.SetNodeSpan(Row: Integer; Span: Integer);
Changes the number of rows a node at cell 0,Row spans.

TMSFMXGrid.GetNodeSpan(Row: Integer): Integer;
Retrieves the number of rows a node spans.

TMSFMXGrid.GetNode(Row: Integer): TCellNode;
Gets the node object used in cell 0,Row.

TMSFMXGrid.OpenNode(Row: Integer);
Opens (expands) all rows within the span of node at cell 0,Row.

TMSFMXGrid.CloseNode(Row: Integer);
Closes (collapses) all rows within the span of node at cell 0,Row.

TMSFMXGrid.OpenAllNodes;
Opens all nodes in the grid.

TMSFMXGrid.CloseAllNodes;
Closes all nodes in the grid.

TTMSFMXContainerGridCell

Inherits from TTMSFMXGridCell and adds the capability of adding a control reference to the cell.

The controls can be set to a container cell via the property: TMSFMXGrid.CellControls[Col,Row: Integer]: TFMXObject

To remove a control from a cell, set TMSFMXGrid.CellControls[Col,Row] to nil.

TTMSFMXBitmapGridCell

Inherits from TTMSFMXGridCell and adds the capability of displaying a bitmap.

Note that a bitmap represents here any graphic format that the FireMonkey framework supports and is not limited to the Windows bitmap formay only. The FireMonkey TBitmap is format agnostic and supports BMP,PNG,GIF,JPEG,ICO files.

The methods to deal with cell bitmaps in the grid are:

TMSFMXGrid.AddBitmap(Col,Row: Integer; AName: string);
Adds a bitmap with name AName from the assigned BitmapContainer to cell Col,Row

TMSFMXGrid.AddBitmap(Col,Row: Integer; ABitmap: TBitmap);
Adds a bitmap instance ABitmap to cell Col,Row

TMSFMXGrid.AddBitmapFile(Col,Row: Integer; AFileName: string);
Adds a bitmap from file AFileName to cell Col,Row

TMSFMXGrid.CreateBitmap(Col,Row: Integer): TBitmap;
Creates a new bitmap instance that is added to cell Col,Row. The bitmap instance can be used to load images from another source.

TMSFMXGrid.AddDataBitmap(Col,Row: Integer);
Adds a data bitmap to cell Col,Row. The bitmap that will be displayed in the cell will depend on the text value in the cell that is used as name in the assigned BitmapContainer

TMSFMXGrid.RemoveBitmap(Col,Row: Integer);
Remove the bitmap from cell Col,Row.

TMSFMXGrid.IsBitmap(Col,Row: integer): boolean;
Returns true when the cell Col,Row contains a bitmap.

TMSFMXGrid.GetBitmap(Col,Row: integer): TBitmap;
Returns the bitmap instance in cell Col,Row.

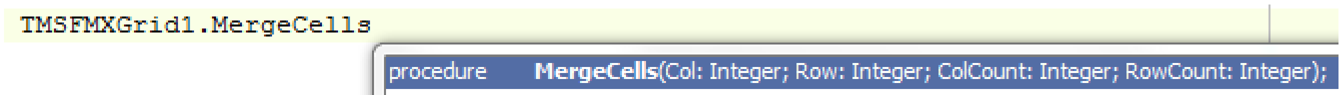
TMSFMXGrid.SetBitmapName(Col,Row: integer; AName: string);
Sets/updates the name of the bitmap referring to the assigned BitmapContainer that was added before with the method AddBitmap()

TMSFMXGrid.GetBitmapName(Col,Row: integer): string;
Returns the name of a bitmap referring to the assigned BitmapContainer.

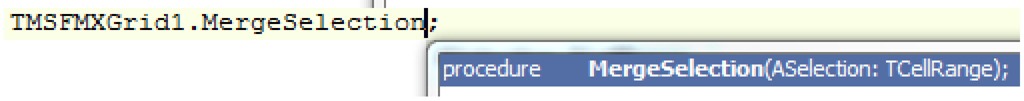
Grid cell merging / splitting

The grid supports merging and splitting cells programmatically as well as with the keyboard.
To merge a range of cells simple call

grid.MergeCells
grid.MergeSelection



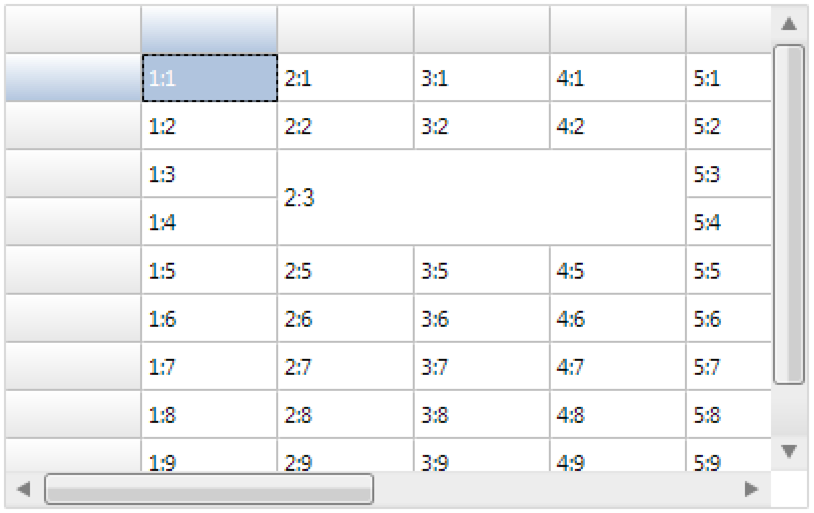
```
TMSFMXGrid1.MergeCells
procedure     MergeCells(Col: Integer; Row: Integer; ColCount: Integer; RowCount: Integer);
```



```
TMSFMXGrid1.MergeSelection;
procedure     MergeSelection(ASelection: TCellRange);
```

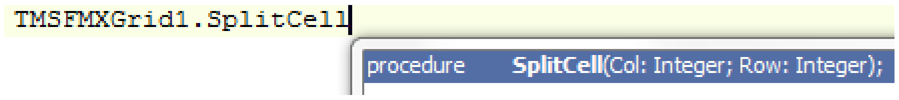
Sample:

```
TMSFMXGrid1.MergeCells(2, 3, 3, 2);
TMSFMXGrid1.MergeSelection(TMSFMXGrid1.CellRange(2, 3, 3, 2));
```



	1:1	2:1	3:1	4:1	5:1
	1:2	2:2	3:2	4:2	5:2
	1:3	2:3			5:3
	1:4				5:4
	1:5	2:5	3:5	4:5	5:5
	1:6	2:6	3:6	4:6	5:6
	1:7	2:7	3:7	4:7	5:7
	1:8	2:8	3:8	4:8	5:8
	1:9	2:9	3:9	4:9	5:9

To split the merged cells, you can use the procedure grid.SplitCell. The parameters passed in the procedure need to be the base cell of the range of merged cells.



```
TMSFMXGrid1.SplitCell
procedure     SplitCell(Col: Integer; Row: Integer);
```

Sample:

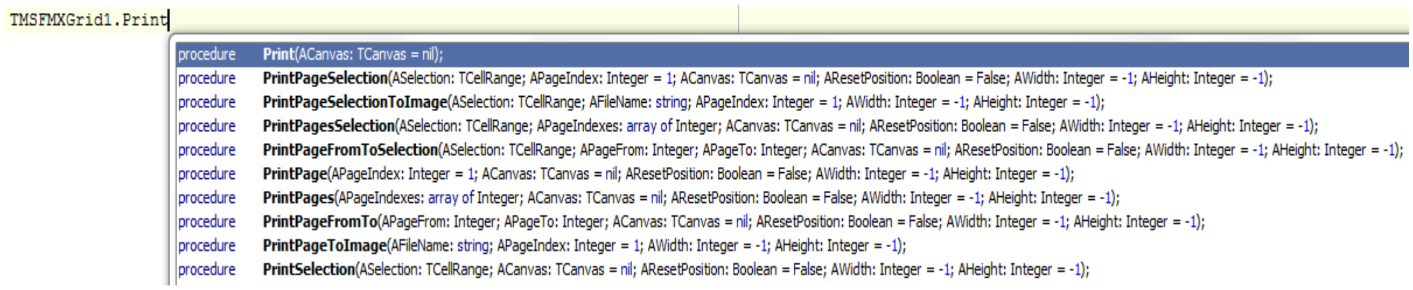
```
 TMSFMXGrid1.SplitCell(2, 3);
```

When enabled via grid.Options.Keyboard.AllowCellMergeShortCuts, the following shortcuts invoke a merge & split of the selected cells:

CTRL + M: merge a selection of cells.
CTRL + S: split a merged cell.

Printing

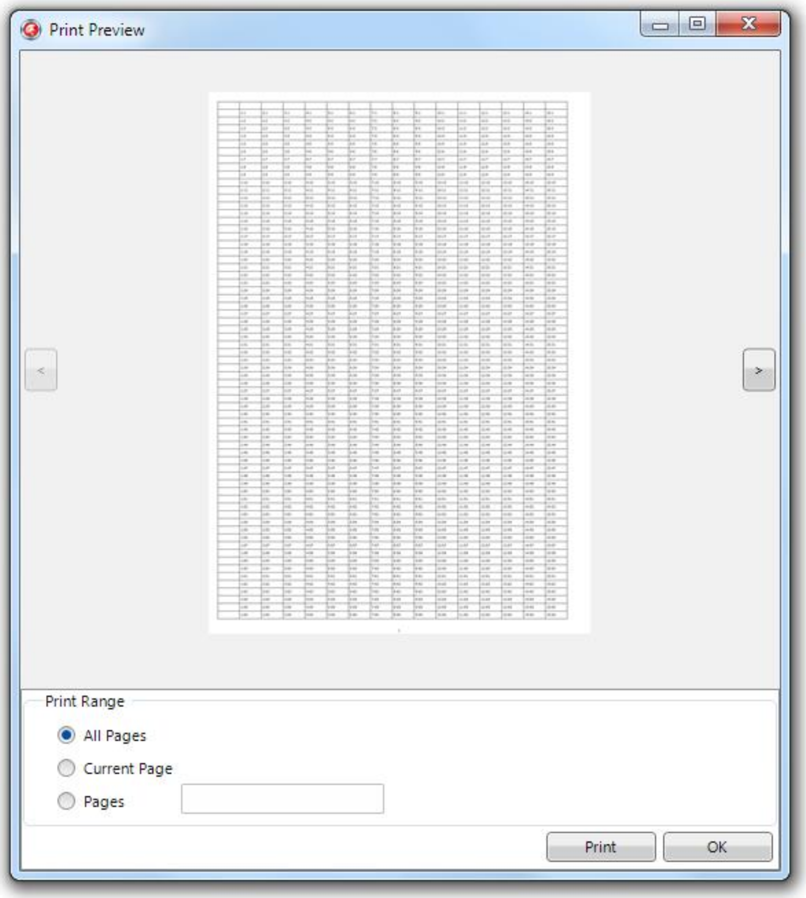
Printing the grid can be done in several ways and with several modes. Below is a list of print procedures that can be used to print the grid. The grid also supports printing on a canvas or an image.



```
TMSFMXGrid1.Print
    procedure   Print(ACanvas: TCanvas = nil);
    procedure   PrintPageSelection(ASelection: TCellRange; APageIndex: Integer = 1; ACanvas: TCanvas = nil; AResetPosition: Boolean = False; AWidth: Integer = -1; AHeight: Integer = -1);
    procedure   PrintPageSelectionToImage(ASelection: TCellRange; AFileName: string; APageIndex: Integer = 1; AWidth: Integer = -1; AHeight: Integer = -1);
    procedure   PrintPagesSelection(ASelection: TCellRange; APageIndexes: array of Integer; ACanvas: TCanvas = nil; AResetPosition: Boolean = False; AWidth: Integer = -1; AHeight: Integer = -1);
    procedure   PrintPageFromToSelection(ASelection: TCellRange; APageFrom: Integer; APageTo: Integer; ACanvas: TCanvas = nil; AResetPosition: Boolean = False; AWidth: Integer = -1; AHeight: Integer = -1);
    procedure   PrintPage(APageIndex: Integer = 1; ACanvas: TCanvas = nil; AResetPosition: Boolean = False; AWidth: Integer = -1; AHeight: Integer = -1);
    procedure   PrintPages(APageIndexes: array of Integer; ACanvas: TCanvas = nil; AResetPosition: Boolean = False; AWidth: Integer = -1; AHeight: Integer = -1);
    procedure   PrintPageFromTo(APageFrom: Integer; APageTo: Integer; ACanvas: TCanvas = nil; AResetPosition: Boolean = False; AWidth: Integer = -1; AHeight: Integer = -1);
    procedure   PrintPageToImage(AFileName: string; APageIndex: Integer = 1; AWidth: Integer = -1; AHeight: Integer = -1);
    procedure   PrintSelection(ASelection: TCellRange; ACanvas: TCanvas = nil; AResetPosition: Boolean = False; AWidth: Integer = -1; AHeight: Integer = -1);
```

Print: Prints the complete grid.
PrintPageSelection: Prints the selection from the grid on a specific page.
PrintPageSelectionToImage: Prints the selection from the grid on a specific page on an image.
PrintPagesSelection: Prints the selection from the grid on multiple selected pages.
PrintPageFromToSelection: Prints the selection from the grid on a specific range of pages.
PrintPage: Prints a specific page from the grid.
PrintPages: Prints a range of pages from the grid.
PrintPageFromTo: Prints a range of pages from the grid.
PrintPageToImage: Prints a specific page from the grid to an image.
PrintSelection: Prints a specific selection from the grid.

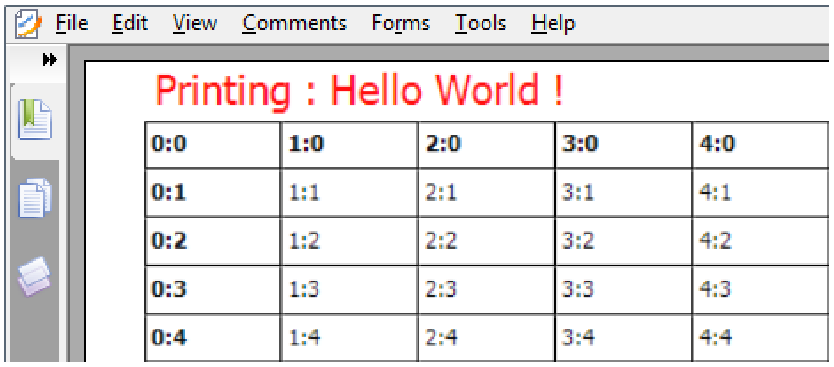
Other than with these procedures, the grid can also be connected to a TTMSFMXGridPrintPreview.
When calling TTMSFMXGridPrintPreview .Execute, the grid will automatically show the first page in the preview window. Navigating can be done with the buttons. The print buttons automatically starts the printing progress of the selected range (all, current page or page range).



Each printing operation is accompied by several events that can be used to track print progress, perform custom drawing on the printer canvas and custom drawing on the printed cell. Events are also triggered when starting new page or ending an existing page as well as drawing the description, title and pagenumber.

Below is a sample that shows the output of the title that is drawn with a different font and color.

```
TMSFMXGrid1.Options.Printing.Description := 'Printing : Hello World!';
TMSFMXGrid1.Options.Printing.DescriptionPosition := ppTopLeft;

procedure TForm1.TMSFMXGrid1PrintDrawDescription(Sender: TObject;
  APageIndex: Integer; ACanvas: TCanvas; var ARect: TRectF; var AText:
string;
  var Allow: Boolean);
begin
  ACanvas.Font.Size := 20;
  ACanvas.Fill.Color := claRed;
end;
```

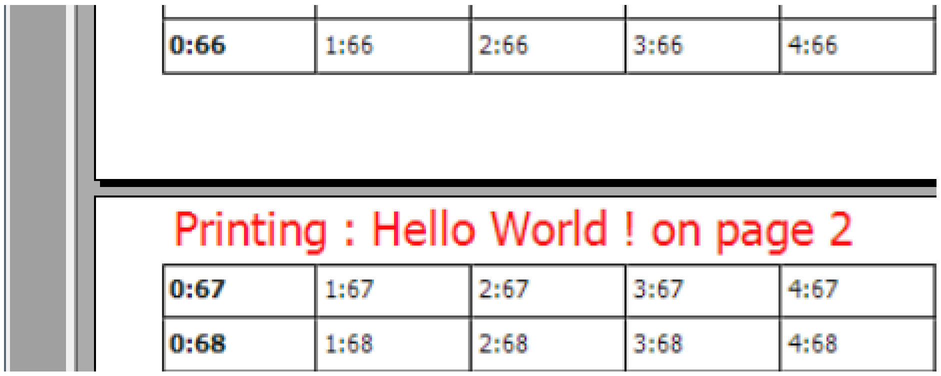


Printing : Hello World !

0:0	1:0	2:0	3:0	4:0
0:1	1:1	2:1	3:1	4:1
0:2	1:2	2:2	3:2	4:2
0:3	1:3	2:3	3:3	4:3
0:4	1:4	2:4	3:4	4:4

To customize the description title or pagenumber per page, you can use the OnPrintNewPage:

```
procedure TForm1.TMSFMXGrid1PrintNewPage(Sender: TObject; APageIndex:
Integer;
  APrinter: TPrinter);
begin
  TMSFMXGrid1.Options.Printing.Description := 'Printing : Hello World
! on page ' + inttostr(APageIndex);
end;
```



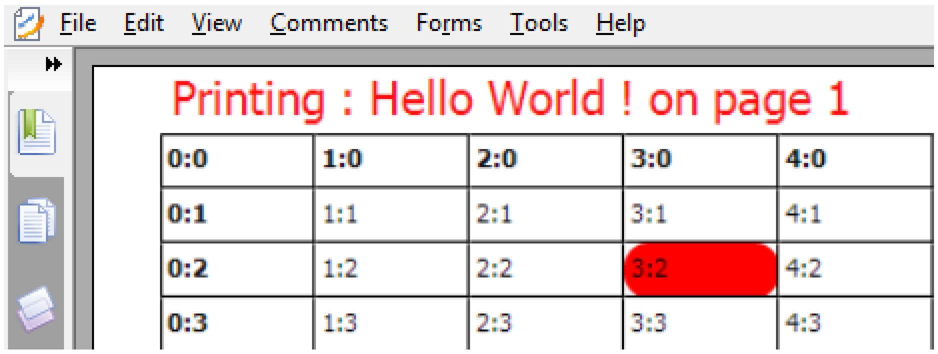
0:66	1:66	2:66	3:66	4:66

Printing : Hello World ! on page 2

0:67	1:67	2:67	3:67	4:67
0:68	1:68	2:68	3:68	4:68

To custom draw a cell, or add additional information to a cell when printing, you can use the event OnPrintDrawCell:

```
procedure TForm1.TMSFMXGrid1PrintDrawCell(Sender: TObject;
  APageIndex: Integer; ACanvas: TCanvas; ARect: TRectF; ACol, ARow:
Integer;
  ACell: TFmxObject; var Allow: Boolean);
begin
  if (ACol = 3) and (ARow = 2) then
  begin
    ACanvas.Fill.Color := claRed;
    ACanvas.FillRect(ARect, 10, 10, AllCorners, 1);
  end;
end;
```



Printing : Hello World ! on page 1

0:0	1:0	2:0	3:0	4:0
0:1	1:1	2:1	3:1	4:1
0:2	1:2	2:2	3:2	4:2
0:3	1:3	2:3	3:3	4:3

The progress of printing can be displayed using the OnPrintProgress event that is passed through as a parameter in percentage

```
procedure TForm1.TMSFMXGrid1PrintProgress(Sender: TObject;
  APageIndex: Integer; APrinter: TPrinter; APrintProgress: Single);
begin
  ProgressBar1.Value := APrintProgress;
end;
```

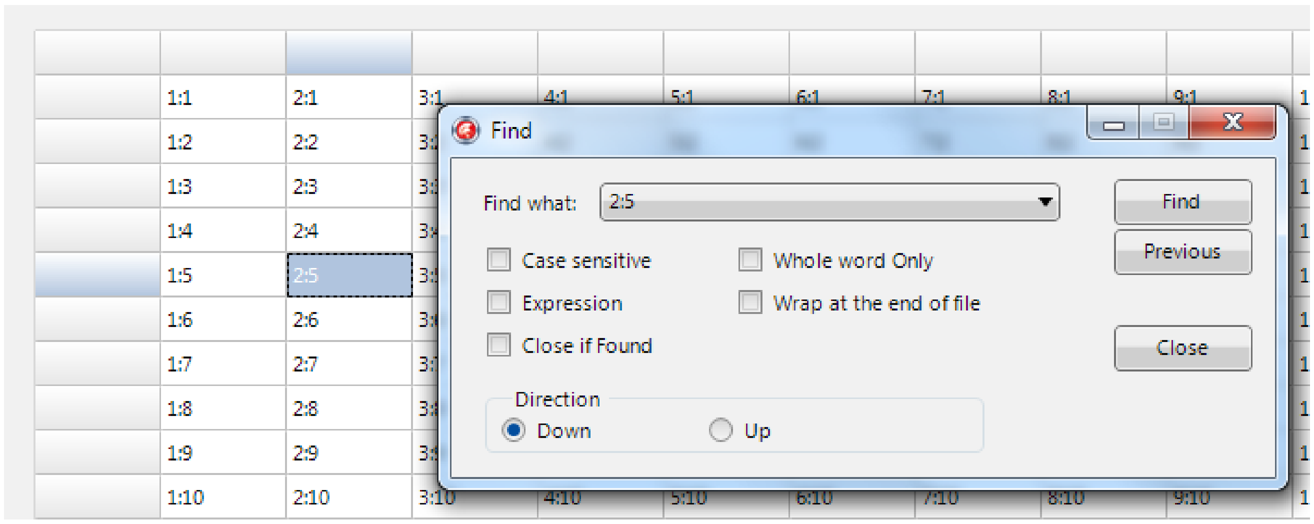
Find & Replace

The grid has built-in support for finding and replacing text with an extensive parameter set and is accompanied by a separate TTMSFMXFindDialog and TTMSFMXReplaceDialog. Below is a sample how to implement this.

Drop a TTMSFMXFindDialog on the form. Add data to the grid and execute the find dialog. When pressing enter in the comboedit control the OnFind event is executed automatically.

In the OnFind event, you can use the grid.Find function to find the text that is entered in the comboedit.

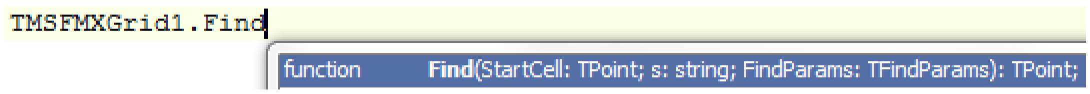
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  TMSFMXFindDialog1.Execute;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  TMSFMXGrid1.ColumnCount := 100;
  TMSFMXGrid1.RowCount := 100;
  TMSFMXGrid1.LinearFill;
end;

procedure TForm1.TMSFMXFindDialog1Find(Sender: TObject);
var
  res: TPoint;
begin
  res := TMSFMXGrid1.Find(Point(1, 1), TMSFMXFindDialog1.FindText,
[fnAutoGoto]);
  if (res.X <> -1) and (res.Y <> -1) then
    TMSFMXFindDialog1.Close;
end;
```



Find what: 2:5

[ ] Case sensitive
[ ] Whole word Only
[ ] Expression
[ ] Wrap at the end of file
[ ] Close if Found

Direction: [✓ Down] Up



```
TMSFMXGrid1.Find
function        Find(StartCell: TPoint; s: string; FindParams: TFindParams): TPoint;
```

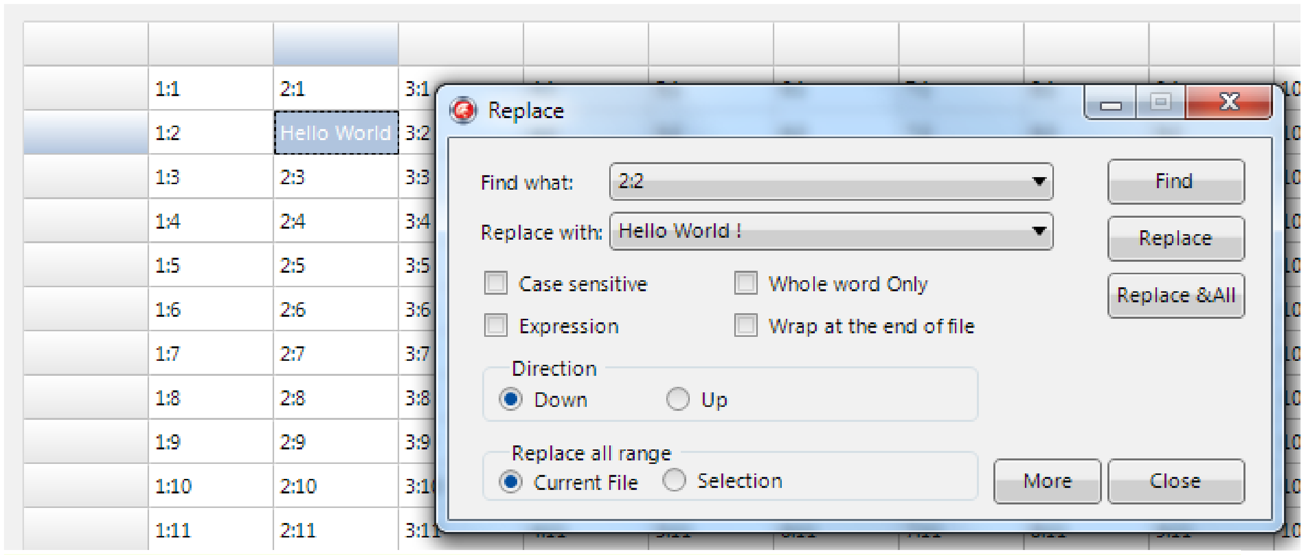
The StartCell parameter is the cell from where the searching must start.
The FindParams option set can be used to search for uppercase, the direction in which to search and many more.

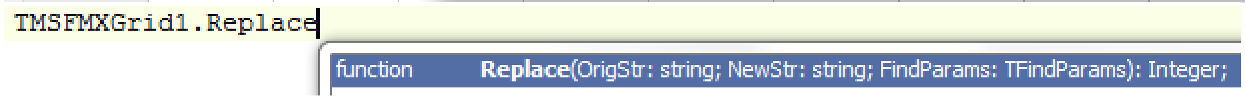
Drop a TTMSFMXReplaceDialog on the form. Add data to the grid and execute the replace dialog. When pressing enter in the comboedit control the OnFind event is executed automatically.

In the OnReplace event, you can use the grid.Replace function to find the text that is entered in the comboedit and replace it with the text entered in the second comboedit.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  TMSFMXReplaceDialog1.Execute;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  TMSFMXGrid1.ColumnCount := 100;
  TMSFMXGrid1.RowCount := 100;
  TMSFMXGrid1.LinearFill;
end;

procedure TForm1.TMSFMXReplaceDialog1Replace(Sender: TObject);
var
  res: Integer;
begin
```

```
  res := TMSFMXGrid1.Replace(TMSFMXReplaceDialog1.FindText,
TMSFMXReplaceDialog1.ReplaceText, [fnAutoGoto, fnMatchFull]);
  if (res <> 0) then
    TMSFMXReplaceDialog1.Close;
end;
```



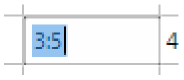


```
TMSFMXGrid1.Replace
function    Replace(OrigStr: string; NewStr: string; FindParams: TFindParams): Integer;
```

Editing

By default the grid supports editing, this can be turned on and off with grid.Options.Editing.Enabled. The default editor is a TTMSFMXEdit which is an control descending from TEdit and that adds a range of new features that offer a more user-friendly experience for grid editing.

To start editing, click on a selected cell to display the inplace editor or press F2 or start typing any key:



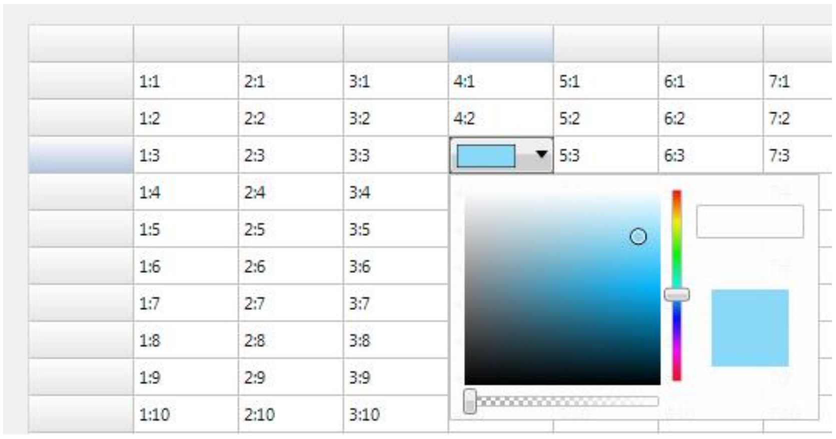
Under grid.Options.Keyboard, grid.Options.Mouse, grid.Options.Editing there are various properties that can be used to customize the way the editing occurs, from direct editing to navigating from edit to edit with the arrow keys, as well as enterkeyhandling to jump to the next row or column, … .

These are the different type of editors available in the grid:

etEdit, etNumericEdit, etSignedNumericEdit, etFloatEdit, etSignedFloatEdit, etUppercaseEdit, etMixedCaseEdit, etLowerCaseEdit, etMoneyEdit, etHexEdit, etAlphaNumericEdit, etValidCharsEdit, etEditBtn, etNumericEditBtn, etSignedNumericEditBtn, etFloatEditBtn, etSignedFloatEditBtn, etUppercaseEditBtn, etMixedCaseEditBtn, etLowerCaseEditBtn, etMoneyEditBtn, etHexEditBtn, etAlphaNumericEditBtn, etValidCharsEditBtn, etComboBox, etComboEdit, etSpinBox, etDatePicker, etDateEdit, etColorPicker, etColorComboBox, etTrackBar, etArcDial, etCustom.

To change an editor type for a specific cell, column or row, implement the OnGetCellEditorType event:

```
procedure TForm1.TMSFMXGrid1GetCellEditorType(Sender: TObject; ACol,
  ARow: Integer; var CellEditorType: TTMSFMXGridEditorType);
begin
  if (ACol = 4) and (ARow = 3) then
    CellEditorType := etColorPicker;
end;
```

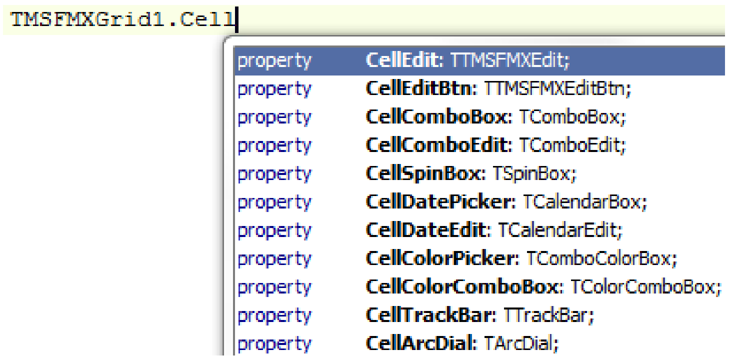


Choosing a color will automatically access the grid.Colors to set the background color of the cell:

Additional settings can be made to the inplace editor via the OnGetCellEditorProperties. Changing the background color and the font color of the default inplace editor:

```
procedure TForm1.TMSFMXGrid1GetCellEditorProperties(Sender: TObject;
ACol,
  ARow: Integer; CellEditor: TFmxObject);
begin
   (CellEditor as TTMSFMXEdit).FontFill.Color := claWhite;
   ((CellEditor as TTMSFMXEdit).FindStyleResource('background') as
TRectangle).Fill.Color := claGreen;
end;
```

The built-in inplace editors can also be accessed separately with:



```
TMSFMXGrid1.Cell
property    CellEdit: TTMSFMXEdit;
property    CellEditBtn: TTMSFMXEditBtn;
property    CellComboBox: TComboBox;
property    CellComboEdit: TComboEdit;
property    CellSpinBox: TSpinBox;
property    CellDatePicker: TCalendarBox;
property    CellDateEdit: TCalendarEdit;
property    CellColorPicker: TComboColorBox;
property    CellColorComboBox: TColorComboBox;
property    CellTrackBar: TTrackBar;
property    CellArcDial: TArcDial;
```

for the editor types listed above.

The possibility exists to use a custom editor. To implement this, etCustom must be set to the var parameter in the OnGetCellEditorType event. For this sample we have used a TTreeView item in a cell that has a modified columnwidth and rowheight:

```
TMSFMXGrid1.ColumnWidths[4] := 150;
TMSFMXGrid1.RowHeights[4] := 100;

procedure TForm1.TMSFMXGrid1GetCellEditorType(Sender: TObject; ACol,
  ARow: Integer; var CellEditorType: TTMSFMXGridEditorType);
begin
  CellEditorType := etCustom;
end;
```

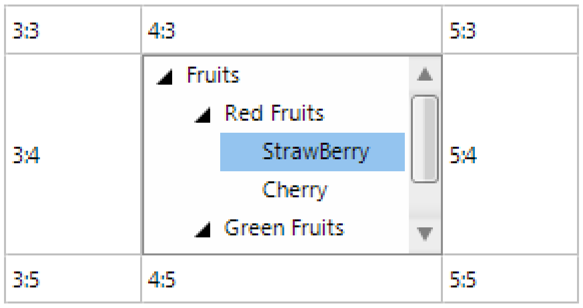
To specify which editor the custom type is, the OnGetCellEditorCustomClassType must be implemented, returning the editor type of choice.

```
procedure TForm1.TMSFMXGrid1GetCellEditorCustomClassType(Sender:
TObject;
  ACol, ARow: Integer; var CellEditorCustomClassType:
TFmxObjectClass);
begin
  CellEditorCustomClassType := TTreeView;
end;
```

Additional properties, items, appearance can be added to the custom editor in the OnGetCellEditorProperties event.

```
procedure TForm1.TMSFMXGrid1GetCellEditorProperties(Sender: TObject;
ACol,
  ARow: Integer; CellEditor: TFmxObject);
var
  tParent, tGroup, tItem: TTreeViewItem;
begin
  tParent := TTreeViewItem.Create(CellEditor);
  tParent.Text := 'Fruits';
  CellEditor.AddObject(tParent);

  tGroup := TTreeViewItem.Create(CellEditor);
  tGroup.Text := 'Red Fruits';
  tParent.AddObject(tGroup);

  tItem := TTreeViewItem.Create(CellEditor);
  tItem.Text := 'StrawBerry';
  tGroup.AddObject(tItem);

  tItem := TTreeViewItem.Create(CellEditor);
  tItem.Text := 'Cherry';
  tGroup.AddObject(tItem);
  tGroup := TTreeViewItem.Create(CellEditor);
  tGroup.Text := 'Green Fruits';
  tParent.AddObject(tGroup);

  tItem := TTreeViewItem.Create(CellEditor);
  tItem.Text := 'Apple';
  tGroup.AddObject(tItem);
  tItem := TTreeViewItem.Create(CellEditor);
  tItem.Text := 'Lime';
  tGroup.AddObject(tItem);
end;
```

When clicking in the cell to start the editor, the treeview is shown.



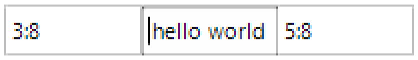
In the OnCellEditDone event, we can set the cell text to the selected item text of the treeview.

```
procedure TForm1.TMSFMXGrid1CellEditDone(Sender: TObject; ACol, ARow:
Integer;
  CellEditor: TFmxObject);
begin
  TMSFMXGrid1.Cells[ACol, ARow] := (CellEditor as
TTreeView).Selected.Text;
end;
```

Intercepting the value from and setting the value in the edit can be done with the OnCellEditGetData, OnCellEditSetData and OnCellEditValidateData.

With the OnCellEditGetData, the data can be intercepted that is passed from the cell to the edit box to set a different text, or append additional text to the cellstring.

```
procedure TForm1.TMSFMXGrid1CellEditGetData(Sender: TObject; ACol,
  ARow: Integer; CellEditor: TFmxObject; var CellString: string);
begin
  CellString := 'hello world !';
end;
```



When the editing is finished the OnCellEditValidateData is called, which can be used to allow / disallow the value to be added in the cell or make additional modifications before the cellstring is allow to be inserted in the cell.

After the validation is true, the OnCellEditSetData is called, which actually inserts the data in the cell. Again, the cellstring can be modified before the string is inserted in the cell.

Editing can be started by calling grid.Edit. The Focused cell will then be set in edit mode and the chosen editor will be shown. To stop editing call grid.StopEdit to persist the value in the cell or call grid.CancelEdit to revert the value back to the value before editing started.

Selection

The selection in the grid is controlled by the property TTMSFMXGrid.SelectionMode. This property determines how cells can be selected in the grid with the mouse and keyboard. The selection varies from single to multiple cells, column and row selections, disjunct selections.

smNone: Hides selection, all other interaction remains active
smSingleCell: Selects a single cell. When changing selection, the previous cell state returns to normal.
smSingleRow: Selects a complete row. When changing selection, the previous row state returns to normal.
smSingleColumn: Selects a complete column. When changing selection, the previous column state returns to normal.
smCellRange: Enables selecting multiple cells. When performing a shift-click, the range between the previous cell and current cell is selected. A range of cells can also be selected when holding and dragging the mouse over the grid.
smRowRange: Enables selecting multiple rows. When performing a shift-click, the range between the previous row and current row is selected. A range of rows can also be selected when holding and dragging the mouse over the grid.
smColumnRange: Enables selecting multiple columns. When performing a shift-click, the range between the previous column and current column is selected. A range of columns can also be selected when holding and dragging the mouse over the grid.
smDisjunctRow: Has the same functionality as smRowRange, and with the ability to distinct select rows with the ctrl key.
smDisjunctColumn: Has the same functionality as smColumnRange and with the ability to distinct select columns with the ctrl key.
smDisjunctCell: Has the same functionality as smCellRange and with the ability to distinct select cells with the ctrl key.

For the selection modes smSingleCell, smSingleRow, smSingleColumn, smCellRange, the property grid.Selection: TCellRange gets or sets the current selected cells. To select for example in the mode smCellRange the cells range 2,2 to 4,4, this can be programmatically set with:

```
TMSFMXGrid1.Selection := CellRange(2,2,4,4);
```

To select a single cell in the mode smSingleCell, the selected cell can be set with:

```
TMSFMXGrid1.Selection := CellRange(3,3,3,3);
```

When the SelectionMode property is smDisjunctRow, two ways are possible to get and set the selected rows:

```
property grid.RowSelect[RowIndex]: Boolean
```

With this property, the selected state of row RowIndex is get or set. A possible way to test for all selected rows as such is:

```
var
  i: integer;
begin
  for i := 0 to TMSFMXGrid1.RowCount - 1 do
  begin
    if TMSFMXGrid1.RowSelect[i] then
      // do something with the selected row
  end;
end;
```

The property grid. RowSelectionCount returns the total number of selected rows in smDisjunctRow selection mode.

An alternative way to get the list of selection rows is by looping through grid. RowSelectionCount and check the index of the selected row returned with grid.

The code to handle this is:

```
var
  i: integer;
begin
  for i := 0 to TMSFMXGrid1.RowSelectionCount - 1 do
  begin
    rowindex := TMSFMXGrid1.SelectedRow[i];
    // do something with the selected row rowindex here
  end;
end;
```

The same applies when the selection mode is smDisjunctColumn with properties grid.ColumnSelect[columnindex]: Boolean, grid.ColumnSelectionCount: integer and grid.SelectedColumn[index]: integer.

Finally, for the selection mode smDisjunctCell, the selection state of a particular cell is returned with grid.CellSelect[col,row: integer]: Boolean;
The total number of selected cells is returned via grid.CellSelectionCount: integer and it is also possible to loop through the list of all selected cells via grid.SelectedCell[Index: integer]: TCell.
In this case, to loop through all selected cells becomes:

```
var
  i: integer;
  c: TCell;
begin
```

```
  for i := 0 to TMSFMXGrid1.CellSelectionCount - 1 do
  begin
    c := TMSFMXGrid1.SelectedCell[i];
    // do something with the selected cell
    TMSFMXGrid1.Cells[c.Col, c.Row] := TMSFMXGrid1.Cells[c.Col, c.Row] + '*';
  end;
end;
```

In combination with the SelectionMode, the grid also supports selection when clicking / dragging on the fixed cells. This can be enabled with grid.Options.Mouse.FixedCellSelection.

fcsAll: Enables clicking on the left top most fixed cell and selects all cells in the grid in combination with smCellRange.
fcsRow: Enables clicking and dragging on the fixed columns / fixed right columns in combination with smSingleRow.
fcsColumn: Enables clicking and dragging on the fixed rows / fixed footer rows in combination with smSingleColumn.
fcsRowRange: Enables clicking and dragging on the fixed columns / fixed right columns in combination with smRowRange.
fcsColumnRange: Enables clicking and dragging on the fixed rows / fixed footer rows in combination with smColumnRange.

If columndragging, rowdragging or sorting is enabled, the fixed cell selection mode is automatically disabled.

Calculations

The grid has built-in methods to perform calculations. Functions are available to perform calculations on all rows or a selected range of rows within a column. These functions generate a result when called. Another type of built-in calculations are column calculations for which the result is displayed in a footer row and for which results are updated as soon as a cell's value changes through editing.

Built-in column calculation functions:

TMSFMXGrid.ColumnSum(ACol: integer; FromRow: integer = -1; ToRow: Integer = -1): Double;
Calculate the sum of values in a column. By default, the sum of all normal cell values is calculated. When the FromRow/ToRow parameters are used, a selected range of rows can be choosen.

TMSFMXGrid.ColumnAvg(ACol: integer; FromRow: integer = -1; ToRow: Integer = -1): Double;
Calculates the average cell value of cells within a column.
TMSFMXGrid.ColumnMin(ACol: integer; FromRow: integer = -1; ToRow: Integer = -1): Double;
Calculates the minimum cell value of cells within a column.
TMSFMXGrid.ColumnMax(ACol: integer; FromRow: integer = -1; ToRow: Integer = -1): Double;
Calculates the maximum cell value of cells within a column.
TMSFMXGrid.ColumnDistinct(ACol: integer; FromRow: integer = -1; ToRow: Integer = -1): Double;
Counts the number of distinct cell valuess within a column.
TMSFMXGrid.ColumnStdDev(ACol: integer; FromRow: integer = -1; ToRow: Integer = -1): Double;
Calculates the number of standard deviation of cell values within a column.
TMSFMXGrid.ColumnCustomCalc(ACol: integer; FromRow: integer = -1; ToRow: Integer = -1): Double;
Performs a custom calculation of values in a column. Calling this method triggers the event OnColumnCalc that should return a result via the var parameter Res.

Built-in automatic column calculations in the footer row:
With the property ColumnCalculation[Col], it can be set what type of calculation result should be displayed in a fixed footer row cell. Possible values are:
ccNone: no result should be displayed in the fixed footer cell
ccSum: column sum should be displayed in the fixed footer cell
ccAvg: column average should be displayed in the fixed footer cell
ccCount: column's row count should be displayed in the fixed footer cell

ccMin: column’s minimum value should be displayed in the fixed footer cell
ccMax: column’s maximum value should be displayed in the fixed footer cell
ccCUSTOM: a custom column calculation result should be displayed in the fixed footer cell
ccDistinct: number of distinct values in the column should be displayed in the fixed footer cell
ccStdDev: column standard deviation should be displayed in the fixed footer cell

When the cell values are updated programmatically, the column calculations can be programmatically updated for one column or for all columns. This can be done with:

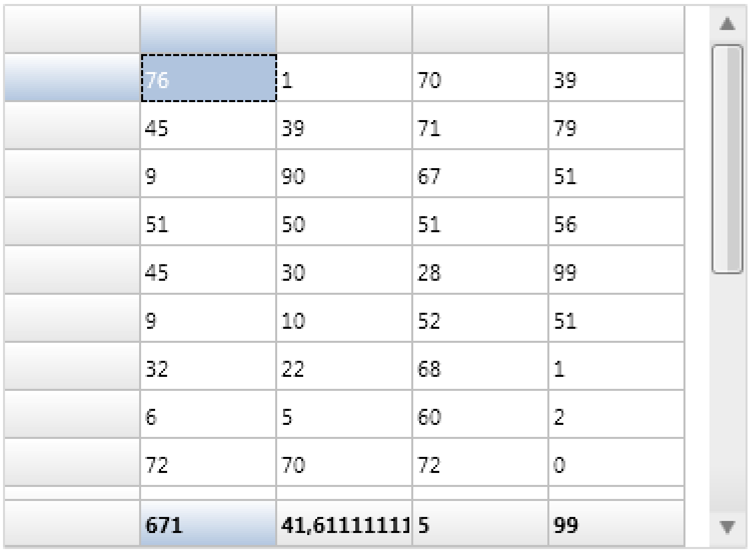
TMSFMXGrid.UpdateCalculations;
TMSFMXGrid.UpdateCalculation(ColumnIndex: integer);

Example:

The grid is initialized with:

```
begin
  TMSFMXGrid1.RowCount := 20;
  TMSFMXGrid1.FixedFooterRows := 1;
  TMSFMXGrid1.RandomFill(false,100);
  TMSFMXGrid1.ColumnCalculation[1] := ccSUM;
  TMSFMXGrid1.ColumnCalculation[2] := ccAVG;
  TMSFMXGrid1.ColumnCalculation[3] := ccMIN;
  TMSFMXGrid1.ColumnCalculation[4] := ccMAX;
  TMSFMXGrid1.UpdateCalculations;
end;
```

and shows upon starting the application:



	76	1	70	39
	45	39	71	79
	9	90	67	51
	51	50	51	56
	45	30	28	99
	9	10	52	51
	32	22	68	1
	6	5	60	2
	72	70	72	0
	671	**41,61111111**	**5**	**99**

When performing editing in the grid cells, the column calculations in the fixed footer row will be automatically updated.

Import / Export

The grid can save and load its data in many different formats explained here:

internal: Saves and loads grid cell data and column widths in a proprietary format
CSV: Saves and loads grid cell data in comma separated file
XLS: Saves and loads grid cell data to an Excel file
XML: Saves the grid cell data to XML file
ASCII: Saves cell data to ASCII file
Fixed: Saves and loads the cell data to fixed length column text files
HTML Saves the cell data to a HTML file
stream: Saves and loads cell data to a stream
RTF: Saves the grid as rich text file

Files

```
procedure SaveToFile(FileName: String; Unicode: boolean = true);
procedure LoadFromFile(FileName: String);
```

SaveToFile saves cell data and column widths to a proprietary file format. LoadFromFile loads cell data and column widths from a proprietary file format. When Unicode = true, the file generated has the BOM marker of a unicode file.

Streams

```
procedure SaveToStream(Stream: TStream);
procedure LoadFromStream(Stream: TStream);
```

SaveToStream saves cell data and column widths to a stream. LoadFromStream loads cell data and column widths from a stream.

Example: copying grid information from grid 1 to grid 2 through a memorystream:

```
var
  ms: TMemoryStream;
begin
  ms := TMemoryStream.Create;
  Grid1.SaveToStream(ms);
  ms.Position := 0; // reset stream pointer to first position
  Grid2.LoadFromStream(ms);
  ms.Free;
end;
```

CSV files

```
procedure SaveToCSV(FileName: String; Unicode: boolean = true);
procedure LoadFromCSV(FileName: String; MaxRows: integer= -1);
procedure AppendToCSV(FileName: String);
procedure InsertFromCSV(FileName: String; MaxRows: integer= -1);
```

SaveToCSV saves cell data to a CSV file. LoadFromCSV loads cell data from a CSV file. AppendToCSV appends cell data to an existing CSV file. InsertFromCSV inserts cell data loaded from the CSV file as extra rows in the grid. Note that LoadFromCSV & InsertFromCSV have a default parameter MaxRows. Without this parameter, all rows in the CSV file are loaded in the grid. When the 2nd parameter MaxRows is used, this sets the maximum number of rows that will be loaded.

Several properties affect the CSV methods:

```
Grid.Options.IO.Delimiter: Char;
```

This specifies the delimiter to use for saving and loading with CSV files. By default the Delimiter is set to #0. With Delimiter equal to #0, an automatic delimiter guess is used to load data from the CSV file. To save to a CSV file, the ; character is used as separator when delimiter is #0. Setting the delimiter to another character than #0 forces the CSV functions to operate with this delimiter only
.

```
Grid.Options.IO.QuoteEmptyCells: Boolean;
```

When true, an empty cell in the CSV file is saved as “”, otherwise no characters are written to the CSV file.

```
Grid.Options.IO.AlwaysQuotes: Boolean;
```

When true, every cell value is saved with prefix and suffix quotes, otherwise quotes are only used if the cell data contains the delimiter character. Note that when the cell data contains quotes, the data is written with doubled quotes to the file.

By default, when loading data in the grid, data is being loaded from the first normal cell, i.e. by default this is cell 1,1 (as by default there is one fixed row and one fixed column). To override this default behavior and make the grid load data at any arbitrary cell, this can be done with the public property

```
TMSFMXGrid.IOOffset: TPoint
```

As such, to start loading data from the first cell 0,0, set

```
TMSFMXGrid.IOOffset := Point(0,0)
```

before calling the LoadFromCSV method.

Fixed column width text files

```
procedure SaveToFixed(FileName: string;positions: TIntList);
procedure LoadFromFixed(FileName:string;positions:TIntList; DoTrim:
boolean = true; MaxRows: integer = -1);
```

SaveToFixed saves cell data and column widths to a text file with fixed column lengths. LoadFromFixed loads cell data and column widths from a text file with fixed column lengths. The TIntList parameter is a list of integer values specifying the character offsets where a column starts in the file.

Example: loading from a fixed file

```
var
  Il: TIntList;
begin
  Il := TIntList.Create(0,0);
  Il.Add(0);    // first column offset
  Il.Add(15);   // second column offset
  Il.Add(30);   // third column offset
  Il.Add(40);   // fourth column offset
  Grid.LoadFromFixed("myfile.txt",il);
  Il.Free;
end;
```

Note that LoadFromFixed has two additional default parameters: DoTrim & MaxRows. When DoTrim is false, spaces before or after words are not removed. Without MaxRows, all rows in the text file are loaded in the grid. When the last parameter MaxRows is used, this sets the maximum number of rows that will be loaded.

HTML files

```
procedure SaveToHTML(FileName: String);
procedure AppendToHTML(FileName: String);
```

SavetoHTML saves the cell data to a HTML file and uses the grid.Options.HTML for various settings that control the export. The cell data is saved to a HTML table. AppendToHTML appends the cell data to an existing HTML file.

XML files

```
procedure SaveToXML(FileName: String; ListDescr,
RecordDescr:string;FieldDescr:TStrings);
```

Saves the cell data in an XML file with following structure:

```
<ListDescr>
<RecordDescr>
<FieldDescr[0]>Cell 0,0</FieldDescr[0]>
<FieldDescr[1]>Cell 1,0</FieldDescr[1]>
<FieldDescr[2]>Cell 2,0</FieldDescr[2]>
</RecordDescr>
<RecordDescr>
<FieldDescr[0]>Cell 0,1</FieldDescr[0]>
<FieldDescr[1]>Cell 1,1</FieldDescr[1]>
<FieldDescr[2]>Cell 2,1</FieldDescr[2]>
</RecordDescr>
</ListDescr>
```

Example:

This code snippet save a grid with 5 columns to XML and uses the text in the column headers as field descriptors in the XML file:

```
var
  sl: TStringList;
  i: integer;
begin
  sl := TStringList.Create;
  for i := 0 to grid.ColCount - 1 do
    sl.Add(grid.Cells[I,0]);
  grid.SaveToXML(„mygrid.xml‟, „xmllist‟, „xmlrecord‟, sl);
  sl.Free;
end;
```

A extra property that is used for exporting to XML file is grid.Options.IO.XMLEncoding that defaults to 'ISO-8859-1'. This property can be used to set a different XML encoding attribute that is saved to the XML file.

ASCII files

```
procedure SaveToASCII(FileName: string);
procedure AppendToASCII(FileName: String);
```

SaveToASCII saves the cell data to an ASCII file, automatically using column widths to fit the widest data in cells available. A difference with fixed column width files is also that SaveToASCII will correctly split cell contents across multiple lines.
AppendToASCII is identical to SaveToASCII, except that it appends the data to an existing file.

XLS files

With the TTMSFMXGridExcelIO component directly reading and writing Excel .XLS files without the need to have Excel installed on the machine is easier than ever.

To use TTMSFMXGridExcelIO for XLS file import or export, follow these steps:

- drop TTMSFMXGrid on a form as well as the component TTMSFMXGridExcelIO
- Assign the instance of TTMSFMXGrid to the Grid property of the TTMSFMXGridExcelIO component
- You can set TTMSFMXGridExcelIO properties to control the Excel file read / write behaviour but in most cases default settings will be ok.
- To import an Excel file, use:

  ```
  TMSFMXGridExcelIO.XLSImport(FileName);
  ```

  or

  ```
  TMSFMXGridExcelIO.XLSImport(FileName,SheetName);
  ```

- To export the contents of TTMSFMXGrid to an XLS file use:

  ```
  TMSFMXGridExcelIO.XLSExport(Filename);
  ```

  or

  ```
  TMSFMXGridExcelIO.XLSExport(FileName,SheetName);
  ```

Properties of TTMSFMXGridExcelIO

Many properties are available in TTMSFMXGridExcelIO to customize importing & exporting of Excel files in the grid.

`AutoResizeGrid: Boolean;`
When true, the dimensions of the grid (ColCount, RowCount) will adapt to the number of imported cells.

`DateFormat: string;`
Sets the format of dates to use for imported dates from the Excel file. When empty, the default system date formatting is applied.

`GridStartCol, GridStartRow: integer;`
Specifies from which top/left column/row the import/export happens

`Options.ExportCellFormats: Boolean;`
When true, cell format (string, integer, date, float) is exported, otherwise all cells are exported as
strings.

`Options.ExportCellMargings: Boolean;`
When true, the margins of the cell are exported

`Options.ExportCellProperties: Boolean;`
When true, cell properties such as color, font, alignment are exported

`Options.ExportCellSizes: Boolean;`
When true, the size of the cells is exported

`Options.ExportFormulas: Boolean;`
When true, the formula is exported, otherwise the formula result is exported

`Options.ExportHardBorders: Boolean;`
When true, cell borders are exported as hard borders for the Excel sheet

`Options.ExportHiddenColumns: Boolean;`
When true, hidden columns are also exported

`Options.ExportHTMLTags: Boolean;`
When true, HTML tags are also exported, otherwise all HTML tags are stripped during export

`Options.ExportImages: Boolean;`
When true, images in the grid are also exported

`Options.ExportOverwrite: Boolean;`
Controls if existing files should be overwritten or not during export

`Options.ExportOverwriteMessage: Boolean;`
Sets the message to show warning to overwrite existing files during export

`Options.ExportPrintOptions: Boolean;`
When true, the print options are exported to the XLS file

`Options.ExportShowGridLines: Boolean;`
When true, grid line setting as set in TAdvStringGrid is exported to the XLS sheet

`Options.ExportShowInExcel: Boolean;`
When true, the exported file is automatically shown in the default installed spreadsheet after export.

`Options.ExportSummaryRowBelowDetail: Boolean;`
When true, summary rows are shown below detail rows in the exported XLS sheet

`Options.ExportWordWrapped: Boolean;`
When true, cells are exported as wordwrapped cells

`Options.ImportCellFormats: Boolean;`
When true, cells are imported with formatting as applied in the XLS sheet

`Options.ImportCellProperties: Boolean;`
When true, cell properties such as color, font, alignment are imported

`Options.ImportCellSizes: Boolean;`
When true, the size of cells is imported

`Options.ImportClearCells: Boolean;`
When true, it will clear all existing cells in the grid before the import is done

`Options.ImportFormulas: Boolean;`
When true, the formula is imported, otherwise only a formula result is imported

`Options.ImportImages: Boolean;`
When true, images from the XLS sheet are imported

`Options.ImportLockedCellsAsReadOnly: Boolean;`
When true, cells that are locked in the XLS sheet will be imported as read-only cells

`Options.ImportPrintOptions: Boolean;`
When true, print settings as defined in the XLS sheet will be imported as grid.PrintSettings

`Options.UseExcelStandardColorPalette: Boolean;`
When true, colors will be mapped using the standard Excel color palette, otherwise a custom palette will be included in the XLS sheet.

`TimeFormat: string;`
Sets the format of cells with a time. When no format is specified, the default system time format is applied.

`XlsStartCol, XlsStartRow: integer;`
Sets the top/left cell from where the import/export should start

Formatting Excel cells when exporting from with TTMSFMXGridExcelIO

By default there is no automatic conversion between the numeric formats in TTMSFMXGrid and Excel since they use different notations. Assume you have the number 1200 in the grid, formatted as "$1,200" .
If you set TTMSFMXGridExcelIO.Options.ExportCellFormat to true, the cell will be exported as the string "$1,200" to Excel. It will look fine, but it will not be a "real" number, and can not be used in Excel formulas. If you set TTMSFMXGridExcelIO.Options.ExportCellFormat to false, the cell will be exported as the number 1200. It will be a real number, that can be added later in Excel, but it will look like "1200" and not "$1,200"

To get a real number that is also formatted in Excel you need to set ExportCellFormat := false, and use the OnCellFormat event in TTMSFMXGridExcelIO, and set the desired format for the cell there.

For example, to have 1200 look like "$1,200" for the numbers in the third column, you could use this event:

```
procedure TMainForm.AdvGridExcelIO1CellFormat(Sender: TAdvStringGrid;
const GridCol, GridRow, XlsCol, XlsRow: Integer; const Value:
WideString;var Format: TFlxFormat);
begin
  if (GridCol = 3) then Format.Format:='$ #,##0';
end;
```

The string you need to write in "Format.Format" is a standard Excel formatting string. It is important to note that this string must be in ENGLISH format, even if your Windows or Excel is not in English. This means that you must use "." as decimal separator and "," as thousands separator, even if they are not the ones in your language.

For information on the available Formatting string in Excel you can consult the Excel documentation, but there is normally a simple way to find out:
Let's imagine that we want to find out the string for a number with thousands separator and 2 decimal places. So the steps are:

- Open an empty Excel file, right click a cell and choose "Format Cells"
- Once the window opens, choose the numeric format you want. Here we will choose a numeric format with 2 decimal places and a thousand separator
- Once we have the format we want, we choose "Custom" in the left listbox. There is no need to close the dialog.

.XLSX files

TMS Grid filters is a component based interface between TTMSFMXGrid and TMS Flexcel to allow to import and export in the .XLSX file format. Free download of the interface components can be found at: http://www.tmssoftware.com/site/advgridfilters.asp

RTF files

With the component TTMSFMXRTFIO, the grid can be exported as a table in rich text formatted file. Drop an instance of TTMSFMXRTFIO on the form and assign the grid to TTMSFMXRTFIO.Grid. Call TTMSFMXRTFIO.ExportRTF(FileName) and a rich text file will be created. Following options are available:

ConvertHTML: Boolean: when true, will convert a cell in the grid that has HTML formatting to a rich text formatted cell. When false, the HTML formatted cell text will be exported as plain text

ExportBackground: Boolean: when true, the background color is exported to the rich text file
ExportBorders: Boolean: when true, grid borders are exported
ExportCellProperties: Boolean: when true, cell properties such as color, font, alignment are exported to the rich text file
ExportHiddenColumns: when true, both visible & hidden columns are exported
ExportImages: when true, images added in cells are exported
ExportMsWordFeatures: when true, the rich text file can contain MS Word specific RTF attributes for a more accurate rendering that might be incompatible with other word processors.
ExportOverwrite: specifies the method to use when exporting to a file that already exists
ExportOverwriteMessage: sets the message for the message box that will be displayed when exporting to a file that already exists
ExportSelectedCells: when true, only selected cells in the grid are exported
ExportShowInWord: when true, opens the generated RTF file in the default application associated with the RTF extension.
Footer: string: sets optional footer rich text
Title: string: sets optional title rich text

Advanced topics on exporting & importing

To apply transformations on cell data for loading and saving it is easy to create a descendent class from TTMSFMXGrid and override the SaveCell and LoadCell methods. In these overridden methods a transformation such as encryption or decryption can be applied. The basic technique is:

```
TEncryptedGrid = class(TTMSFMXGrid)
protected
  function SaveCell(ACol,ARow: Integer):string; override;
  procedure LoadCell(ACol,ARow: Integer; Value: string); override;
end;

function TEncryptedGrid.SaveCell(ACol,ARow: Integer): string;
begin
  Result := Encrypt(GridCells[ACol,ARow]);
end;

procedure TEncryptedGrid.LoadCell(ACol,ARow: Integer; Value: string);
begin
  GridCells[ACol,ARow] := Decrypt(Value);
end;
```

As such, when using methods like SaveToCSV, SaveToASCII, … the information will be exported in encrypted format automatically.

Sorting

The grid supports 2 types of sorting: normal sorting and indexed sorting. In normal sorting mode, the grid sorts the data ascending or descending on a specified column. In indexed sorting multiple columns can be marked and sorted ascending or descending. In both modes, the sorted column is marked with a triangle that is displayed with a number in indexed mode. The sort column can be set programmatically or by clicking on a fixed column header cell.

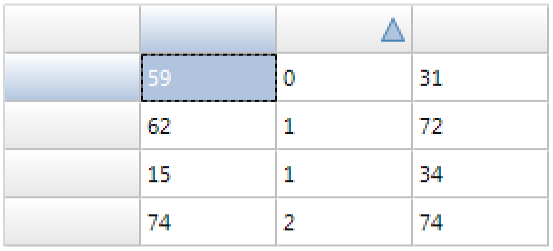
By default sorting is disabled. Enabling sorting can be done by setting the mode:

```
grid.Options.Sorting := gsmNormal;
```

or

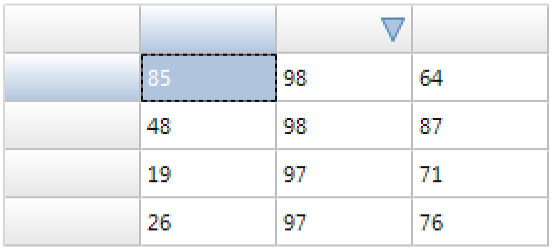
```
grid.Options.Sorting := gsmIndexed;
```

In normal mode, when clicking on a column, or setting grid.SortColumn := 3 an indicator appears that indicates a column is sorted in ascending order.



		△	
	59	0	31
	62	1	72
	15	1	34
	74	2	74

Clicking the same column again, changes the order to descending.



		▽	
	85	98	64
	48	98	87
	19	97	71
	26	97	76

Setting the sorting mode to gsmIndexed will show a yellow triangle with an index number instead of a blue rectangle.

By default, the grid will automatically guess the data format of a cell to determine the compare method to use. It will detect regular strings, numbers and dates. Additional control over the compare methods to use per column is available via the event OnSortFormat:

```
TTMSFMXGridSortFormatEvent = procedure(Sender: TObject; Col: Integer;
var SortFormat: TSortFormat; var APrefix, ASuffix: string) of object;
```

The TSortFormat type is:

ssAutomatic: try to automatically guess the column data type to control the compare method
ssAlphabetic: cells contain text, compare with case sensitivity
ssAlphabeticNoCase: cells contain text, compare without case sensitivity
ssNumeric: cells in column contain a number, sort based on numeric comparisons
ssDate: cells in column contain a date, sort based on date comparisons
ssHTML: cells in column contain HTML formatted text, compare cells based on text without HTML tags
ssCheckBox: cells in column contain checkboxes, using compare of boolean values
ssCustom: a custom compare will be performed via the event OnCustomCompare
ssRaw: a custom compare will be performed via the event OnRawCompare.

The parameters APrefix, ASuffix allow to set a text as either prefix or suffix that will be ignored in the comparison. If the cell text is for example displaying a currency symbol like: 125.00$, by setting ASuffix to '$', the comparison can be based just on the numeric data 125.00.

Custom sorts
Two events, OnCustomCompare and OnRawCompare are used to allow implementing custom compare routines when the sort format style is specified as ssCustom or ssRaw.

The OnCustomCompare is triggered for each compare of two string values and expects the result to be set through the Res parameter with values:

-1: Str1 < Str2
0: Str1 = Str2
1: Str1 > Str2

The OnRawCompare event is defined as:

```
TRawCompareEvent = procedure(Sender:TObject; ACol,Row1,Row2: Integer;
var Res: Integer) of object;
```

It allows comparing grid cells [ACol,ARow1] and [ACol,ARow2] in any custom way and returning the result in the Res parameter in the same way as for the OnCustomCompare event.

Example: comparing cell objects instead of cell text with OnRawCompare

As for each cell, an object can be assigned with the grid.Objects[Col,Row]: TObject property, it is easy to associate a number with each cell through:

```
Grid.Cells[Col,Row] := „I am text‟; // cell text
Grid.Objects[Col,Row] := TObject(1234); // associated number
```

Through the OnRawCompare event, a sort can be done on this associated number instead of the cell text.

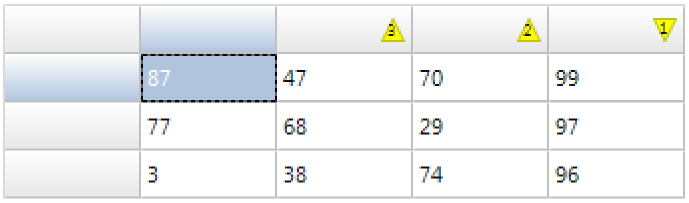
```
procedure TTMSFMXGridOnRawCompare(Sender: TObject; ACol, Row1, Row2:
Integer; var Res: Integer);
var
  c1,c2: Integer;
begin
  c1 := integer(TMSFMXGrid1.Objects[ACol,Row1]);
  c2 := integer(TMSFMXGrid1.Objects[ACol,Row2]);
  if (c1 = c2) then
    Res := 0
  else
  if (c1 > c2) then
     Res := 1
  else
     Res := -1;
end;
```

Finally, two events that can be useful for sorting are: OnCanSortColumn and OnCellSortClick. The event OnCanSortColumn is triggered when a fixed column header cell is clicked just before an actual sort is performed. The event informs about the column clicked and passes the parameter Allow. By setting it to false, no actual sort is performed. The event OnCellSortClick is triggered after the sort on a specific column is done. While sorts in average sized grids is mostly instantanous, note that these two events could be used to set for example the mouse cursor as wait cursor from the OnCanSortColumn event and reset it to default cursor from the OnCellSortClick event when sorting very large grids.

To perform indexed sorting programmatically, add the columns that will be used as sort criteria to the grid.SortIndexes list. The first added column to the list is the first sort criteria, the second column added is the second criteria etc. For each sort column added, the sort order can be set with the second parameter of the AddIndex method. Call grid.SortIndexed after filling the SortIndexes list to perform the sorting.

```
  TMSFMXGrid1.SortIndexes.Clear;
  TMSFMXGrid1.SortIndexes.AddIndex(3, sdAscending);
  TMSFMXGrid1.SortIndexes.AddIndex(4, sdDescending);
  TMSFMXGrid1.SortIndexed;
```

To perform sorting on a single column click on the fixed column header of choice.To perform indexed sorting from the UI, click the first fixed column header cell to set the primary sort column and after this, hold shift and click on the additional columns a sort criteria needs to be set for. A regular click removes all the indexes and sets the primary sort column again.



		▲3	▲2	▼1
	87	47	70	99
	77	68	29	97
	3	38	74	96

Note that when the grid is grouped, the sorting is automatically performed within groups. Sorting within groups can be based on a single column or can use indexed sorting as well. If it is needed that groups itself are resorted, perform an ungroup, perform the sort wanted and then regroup. To programmatically perform a sort in a grouped grid, call TMSFMXGrid.SortGrouped(Column, Direction). To programmatically perform an indexed sort in a grouped grid, add the indexes of columns to sort on first to the SortIndexes collection and then call TMSFMXGrid.SortGroupedIndex;

```
TMSFMXGrid1.SortIndexes.Clear;
TMSFMXGrid1.SortIndexes.AddIndex(5, sdDescending);
TMSFMXGrid1.SortIndexes.AddIndex(2, sdAscending);
TMSFMXGrid1.SortGroupedIndexed;
```

Grouping

TTMSFMXGrid has built-in single level automatic grouping and grouped sorting. This makes it easy to add grouping features with a few lines of code. Grouping means that identical cells within the same column are removed and shown as a grouping row for the other cells in the rows.

Example:

United States	New York	205000
United States	Chicago	121200
United States	Detroit	250011
Germany	Köln	420532
Germany	Frankfurt	122557
Germany	Berlin	63352

Grouped on the first column this becomes:

- United states	
New York	205000
Chicago	121200
Detroit	250011
- Germany	
Köln	420532
Frankfurt	122557

Berlin	63352

Grouped sorting on the first column becomes:

- United states	
Chicago	121200
Detroit	250011
New York	205000
- Germany	
Berlin	63352
Frankfurt	122557
Köln	420532

This is an overview of the grouping methods:

```
procedure Group(ColIndex:integer);
procedure UnGroup;
```

The Group method groups based on the column ColIndex. It automatically adds the expand / contract nodes. When expand / contract nodes are available, the normal sort when a column header is clicked changes to inter group sorting.
Note that the column for grouping can only start from column 1, since column 0 is the placeholder for the expand / contract nodes.

To undo the effect of grouping, the UnGroup method can be used.

Example: loading a CSV file, applying grouping and performing a grouped sort

```
// loading CSV file in normal cells
TMSFMXGrid1.LoadFromCSV('cars.csv');
TMSFMXGrid1.ColWidths[0] := 20;

// do grouping on column 1
TMSFMXGrid1.Group(1);

// apply grouped sorting on (new) column 1
TMSFMXGrid1.SortGrouped(1, sdAscending);
```

When a grouped view is no longer necessary, it can be removed by:

```
TMSFMXGrid.UnGroup;
```

Extra grouping features

Some extra capabilities for more visually appealing grouping can be set through the property grid.Options.Grouping. Through this property it can be enabled that group headers are automatically set in a different color and that cells from a group header are automatically merged. In addition, a group can also have a summary line. A summary line is an extra row below items that belong to the same group. This summary line can be used to put calculated group values in. The color for this summary line can also be automatically set as well as cell merging performed on this. See the grid.Options.Grouping description for all details.

Group calculations

TTMSFMXGrid has built-in function to automatically calculate group sums, average, min, max, count. The group results are set in the group header row if no summary row is shown, otherwise the group summary row is used by default. Group calculations are performed per column.

Available functions:

```
grid.GroupSum(AColumn: Integer);
```
Calculates column sums per group

```
grid.GroupAvg(AColumn: Integer);
```
Calculates column averages per group

```
grid.GroupMin(AColumn: Integer);
```
Calculates column minimum per group

```
grid.GroupMax(AColumn: Integer);
```
Calculates column minimum per group

```
grid.GroupCount(AColumn: Integer);
```
Calculates number of rows in a group for each group

```
grid.GroupDistinct(AColumn: Integer);
```
Calculates number of distinct rows in a group for each group

```
grid.GroupStdDev(AColumn: Integer);
```
Calculates standard deviation of values in column AColumn within a group for each group

```
grid.GroupCustomCalc(AColumn: Integer);
```
Allows to perform a custom calculation of group data with the event OnGroupCalc

If there is a need for a special group calculation that is not available in the standard group calculation functions, the method grid.GroupCustomCalc can be used. For each group in the grid, this will trigger the event

```
grid.OnGroupCalc(Sender: TObject; ACol, FromRow, ToRow: Integer; var
Res: Double);
```

The meaning of the parameters is: ACol : column to perform calculation for FromRow: first row in the group ToRow: last row in the group Res: variable parameter to use to set the result

In this sample, the grid is initialized with random number, is grouped on column 1 and for the first column in the grouped grid the standard deviation is calculated:

```
procedure TForm1.TMSFMXGrid1GroupCalc(Sender: TObject; ACol, FromRow,
ToRow: Integer; var Res: Double);
var
   i: integer; d, m, sd: double;
begin
  // calculate mean
m := 0;
for i := FromRow to ToRow do
begin
  m := m + TMSFMXGrid1.Floats[ACol,i];
end;
m := m / (ToRow - FromRow + 1);

// calculate standard deviation
sd := 0;
for i := FromRow to ToRow do
begin
  sd := sd + sqr(TMSFMXGrid1.Floats[ACol,i] - m);
end;
sd := sd / (ToRow - FromRow);
Res := sqrt(sd);
end;
```

Column persistence

The grid offers various helper functions to deal in code with moving columns & sizing columns from the UI and persisting column width, column position and column visibility.

Following methods are available for this:

procedure SetColumnOrder;

It is important to note that all column movement tracking is done with respect to a reference column ordering. The reference column ordering is assumed to be the order of the columns when grid.SetColumnOrder is called. This internally initializes the sequence of the columns as the first column being column 0, the 2nd column being column 1, etc… All further column moving will be tracked against this ordering. As such, call grid.SetColumnOrder when the grid is initialized with data and the required grid.ColumnCount is set.

procedure ResetColumnOrder;

Calling grid.ResetColumnOrder moves the columns back to the initial sequence, i.e. the sequence when grid.SetColumnOrder was called. Irrespective of how the user moved columns via column drag & drop, it will reset the grid to the original column sequence. This will not affect the column widths.

function ColumnStatesToString: string;

This returns the states of each column as a string. This string can be easily stored in a registry or INI file or database for example. This string represents the current column ordering, the widths of the columns and the column visibility. The states of the columns returned via ColumnStatesToString is the state relative to the reference order determined at the time grid.SetColumnOrder was called. As such, a typical scenario is to call grid.ColumnStatesToString before the application closes and store this. With this stored value, the sequence and width of the columns can be restored to the state when the user left the application when it is restarted.

procedure StringToColumnStates(States: string);

Assuming the grid is in reference column order (if not call grid.ResetColumnOrder), a previously stored state of columns can be restored by calling StringToColumnStates with the string that represents the state as parameter.

function Columnposition(ACol: integer): integer;

When the reference column order is set, the function ColumnPosition() can be used to get the position of a specific column after the user moved columns around with drag & drop.

function ColumnAtPosition(APosition: integer): integer;

When the reference column order is set, the function ColumnAtPosition can be used to return the index of the column in its reference order that is at a specific position after a user moved the columns around.

Columns

The Columns collection manages designtime and runtime grid cell layout, types and behaviour as well as cell interaction capabilities such as sorting and editing. This behaviour is default and is controlled with the public UseColumns property.

When using combinations of dynamically created checkboxes at runtime and checkbox columns at designtime with the columns collection the preference is given to the dynamically created checkboxes.

Below are the properties that can be used to configure a grid column. All properties that are related to appearance / layout of a cell are applied only to normal cells. Other cell types are configured dynamically through one of the events that can be used to modify the cell layout.

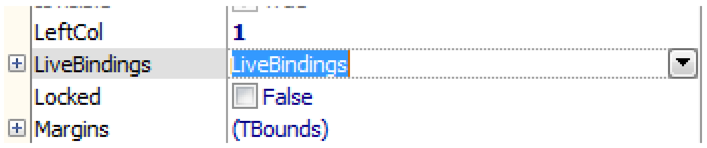
BorderColor: TAlphaColor: Border color of a column grid cell in normal state.
BorderWidth: Single: Border width of a column grid cell in normal state.
Color: TAlphaColor: Color of a column grid cell in normal state.
ColumnType: TTMSFMXGridColumnType: Identifies the type of the column. A column can be configured to show checkbox, radiobutton, button or progressbar cell types. The ColumnType property has a default value which is a normal grid cell type.
ComboItems: TStringList: Used in combination with the Editor property. The ComboItems property is a stringlist that is assigned to the CellComboBox internally used for editing, when the editor type is etComboBox or etComboEdit.
Editor: TTMSFMXGridEditorType: The editor type used to define the inplace editor that is used for editing and is identical to the editor type retrieved through the OnGetCellEditorType. Used in combination with the ComboItems property in case of etComboBox or etComboEdit.
Fixed: Boolean: Sets the complete column as a fixed column. The cell type will be modified to a fixed cell type and therefore all layout properties such as Color, BorderColor and FontColor are ignored.
FontColor: TAlphaColor: The color of the font of a column grid cell in normal state.
Font: TFont: The font of a column grid cell in normal state.
HorzAlignment: TTextAlign: The horizontal alignment of a column grid cell text in normal state.
ID: String: A unique identifier for each column to make sure each column can be accessed with this unique identifier after a column has been swapped, inserted or deleted.
ReadOnly: Boolean: Sets the complete column readonly. The cells for that column remain normal cell types but are not editable.
SortFormat: TSortFormat: The sorting format type of the column when sorting is applied ("Sorting" chapter). The property values can be set to ssAutomatic which will automatically identify the content of the column cells, a specific value such as ssAlphabetic,

ssAlphabeticNoCase, ssNumeric, ssDate, ssHTML, ssCheckBox, ssRaw (OnRawCompare) or ssCustom (OnCustomCompare).
SortSuffix: string: A sorting suffix used for additional sorting customization.
SortPrefix: string: A sorting prefix used for additional sorting customization.
Tag: integer: A second unique identifier that can be used in a similar way as the ID property.
VertAlignment: TTextAlign: The vertical alignment of a column grid cell text in normal state.
WordWrap: Boolean: The wordwrap of a column grid cell text in normal state.

LiveBindings

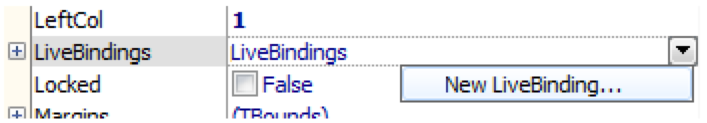
When reading the documentation in the "LiveBindings in RAD Studio" you will notice that LiveBindings is not limited to DataBase support. There is also support for binding various properties of the Grid to other controls. Below is a sample that binds the TrackBar position to the Grid rotationAngle.

Drop a new Grid and a TrackBar component on the form. When dropping a Grid on the form you will notice a LiveBindings property.



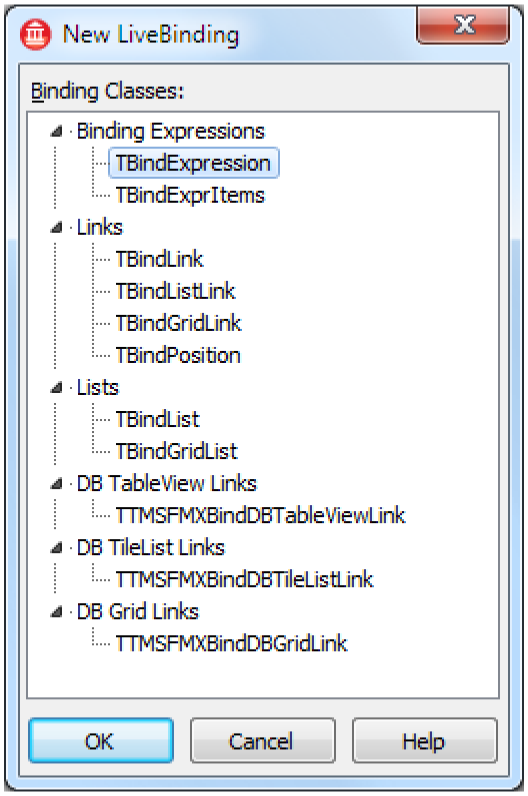
LeftCol	1
⊞ LiveBindings	LiveBindings
Locked	☐ False
⊞ Margins	(TBounds)

To create a new LiveBinding, you can either click on the arrow and select "New LiveBinding…" or click directly on "New LiveBinding…" at the bottom of the object inspector.

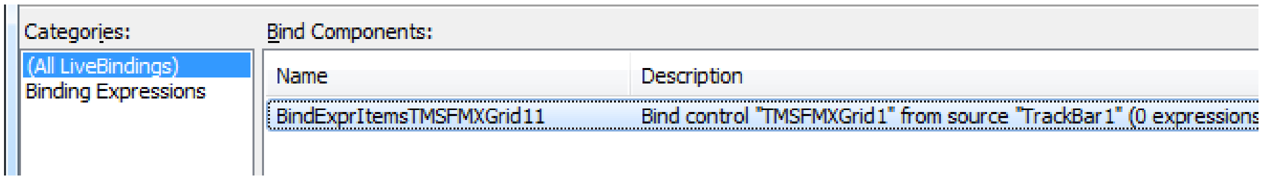


LeftCol	1
⊞ LiveBindings	LiveBindings
Locked	☐ False

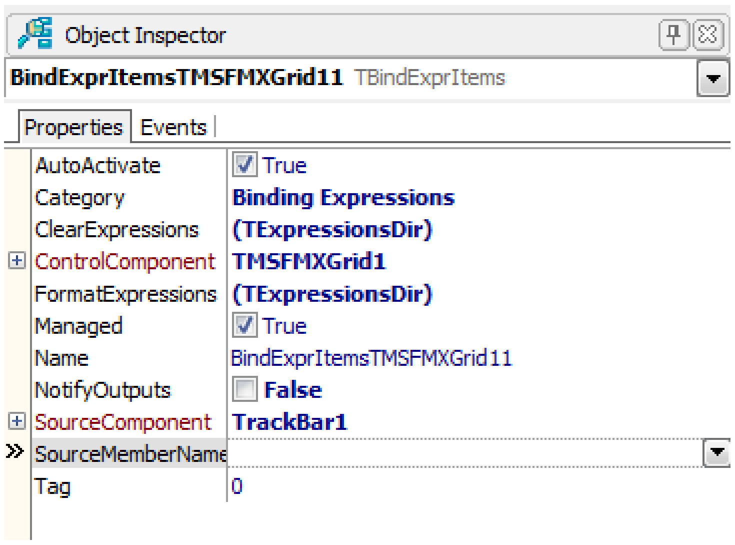
This action will automatically drop a BindingsList component on the form and will show the BindingsList editor window.



Select the TrackBar component and add a new TBindExprItems expression. In the BindingsList component you see the TBindExprItems component listed.



Name	Description
BindExprItemsTMSFMXGrid11	Bind control "TMSFMXGrid1" from source "TrackBar1" (0 expressions

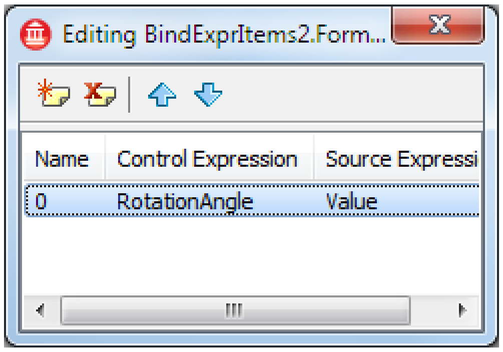

Object Inspector

BindExprItemsTMSFMXGrid11 TBindExprItems

Properties | Events

AutoActivate	✓ True
Category	Binding Expressions
ClearExpressions	(TExpressionsDir)
ControlComponent	TMSFMXGrid1
FormatExpressions	(TExpressionsDir)
Managed	✓ True
Name	BindExprItemsTMSFMXGrid11
NotifyOutputs	False
SourceComponent	TrackBar1
SourceMemberName	
Tag	0

Point the SourceComponent to the TrackBar and the ControlComponent to the TMSFMXGrid1. Start the FormatExpressions editor by double-clicking on the “(TExpressionsDir)” label. Click on the add button to add a new format expression and fill in the Binding properties. The ControlComponent is set to TMSFMXGrid1 and we want the RotationAngle to be modified. Fill in RotationAngle in the Control Expression field. For the Source Expression which is linked with the SourceComponent we fill in “Value”.


Editing BindExprItems2.Form...

Name	Control Expression	Source Expressi
0	RotationAngle	Value

Now there is one step left to implement before the application is ready. When dragging the slider of the TrackBar, the RotationAngle of the TMSFMXGrid component will be unaffected. This is because the TMSFMXGrid did not receive a notice from the BindingsList component, therefore we need to notify the bind component that the value of the TrackBar has changed.

This is done by implementing the OnChange event and notifying the BindingsList component:

```
var
  FNotifying: Integer;
```

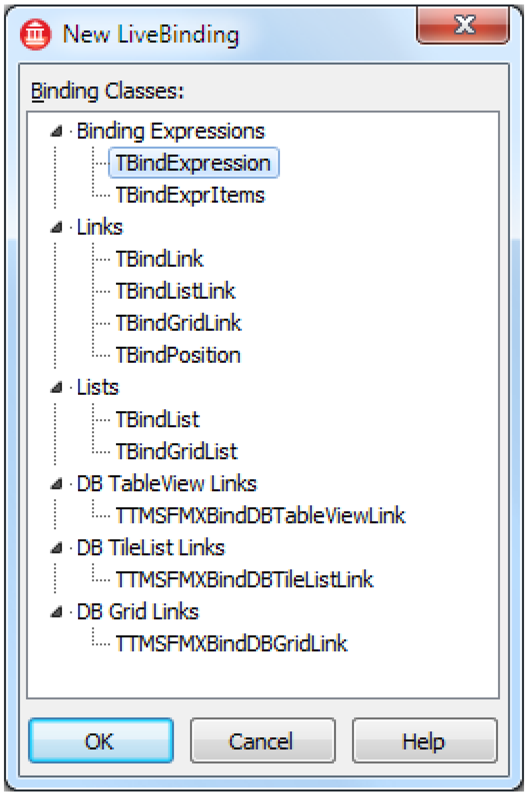
```
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  // Some controls send notifications when setting properties,
  // like TTrackBar
  if FNotifying = 0 then
  begin
    Inc(FNotifying);
    // Send notification to cause expression re-evaluation of
dependent expressions
    try
      BindingsList1.Notify(Sender, '');
    finally
      Dec(FNotifying);
    end;
  end;
end;
```

Now when dragging the slider of the TrackBar, the BindingsList is notified and the TMSFMXGrid rotates according to the Value of the TrackBar. Multiple bindings can be made and are triggered simultaneously due to the Notify procedure of the BindingsList.

The Grid supports displaying fields and records as well as editing and navigating. This can be done manually or automatically.

Manually

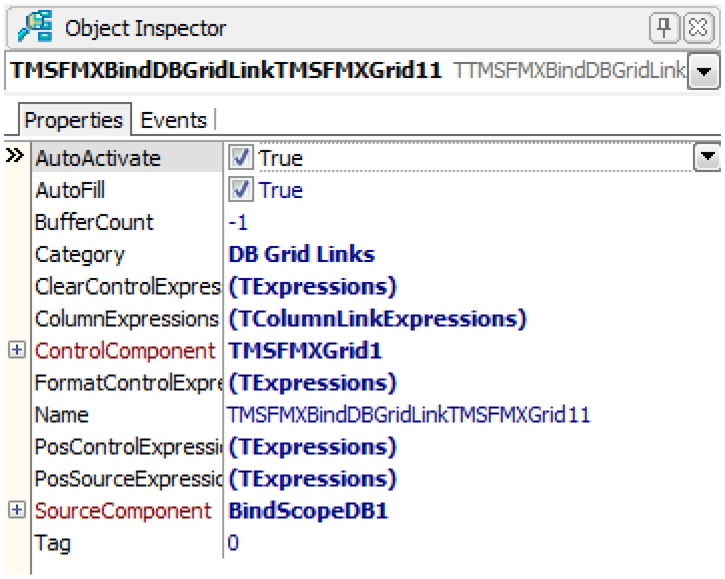
In the BindingsList editor window you will notice that a new category is added for the Grid: "DB Grid Links". The link "TTMSFMXBindDBGridLink" can be used to connect to a DataSource through a BindScope. This link is designed to work specifically with DataBase connections.


New LiveBinding

Binding Classes:

- Binding Expressions
  - TBindExpression
  - TBindExprItems
- Links
  - TBindLink
  - TBindListLink
  - TBindGridLink
  - TBindPosition
- Lists
  - TBindList
  - TBindGridList
- DB TableView Links
  - TTMSFMXBindDBTableViewLink
- DB TileList Links
  - TTMSFMXBindDBTileListLink
- DB Grid Links
  - TTMSFMXBindDBGridLink

The PosControlExpression and PosSourceExpression are used to link the position in the dataset to the selected row and vice versa. This is automatically generated when using a TTMSFMXBindDBGridLink component.


Object Inspector

TMSFMXBindDBGridLinkTMSFMXGrid11 TTMSFMXBindDBGridLink

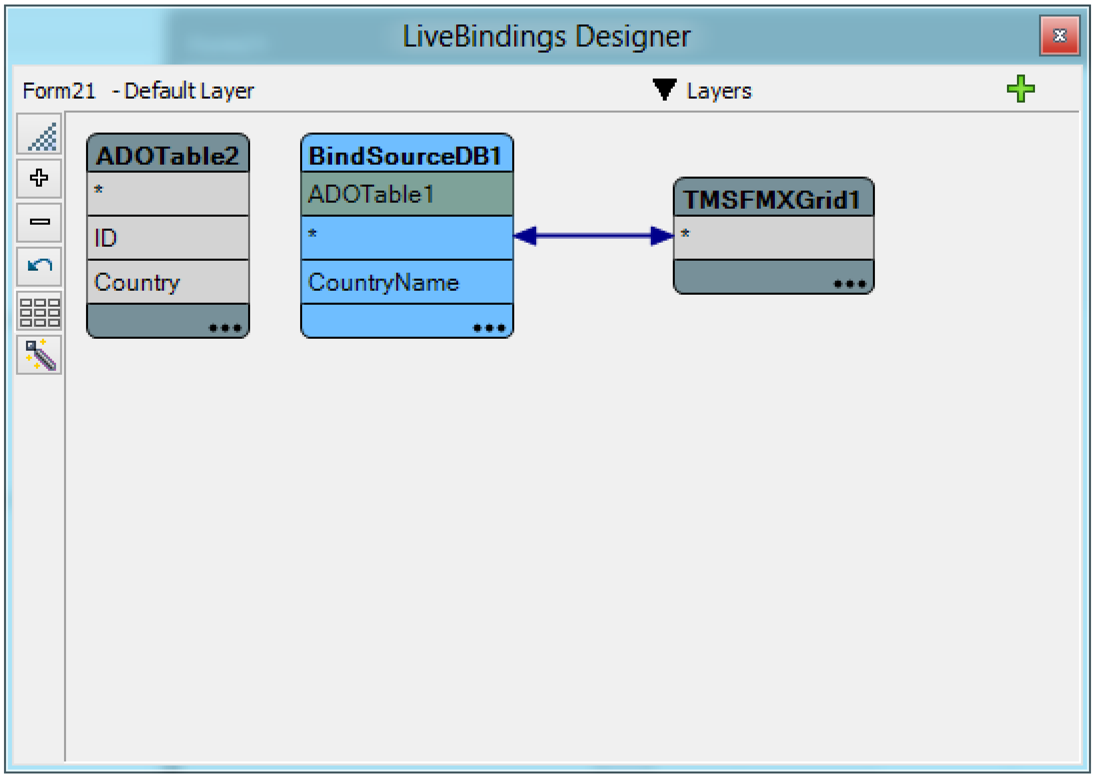
AutoActivate	True
AutoFill	True
BufferCount	-1
Category	DB Grid Links
ClearControlExpres	(TExpressions)
ColumnExpressions	(TColumnLinkExpressions)
ControlComponent	TMSFMXGrid1
FormatControlExpre	(TExpressions)
Name	TMSFMXBindDBGridLinkTMSFMXGrid11
PosControlExpressi	(TExpressions)
PosSourceExpressic	(TExpressions)
SourceComponent	BindScopeDB1
Tag	0

In the object inspector, after selecting the TTMSFMXBindDBGridLink component you will notice a ColumnExpressions property. This is because the implementation inherits from the standard TBindGridLink and is necessary to link Database fields to columns in the Grid.

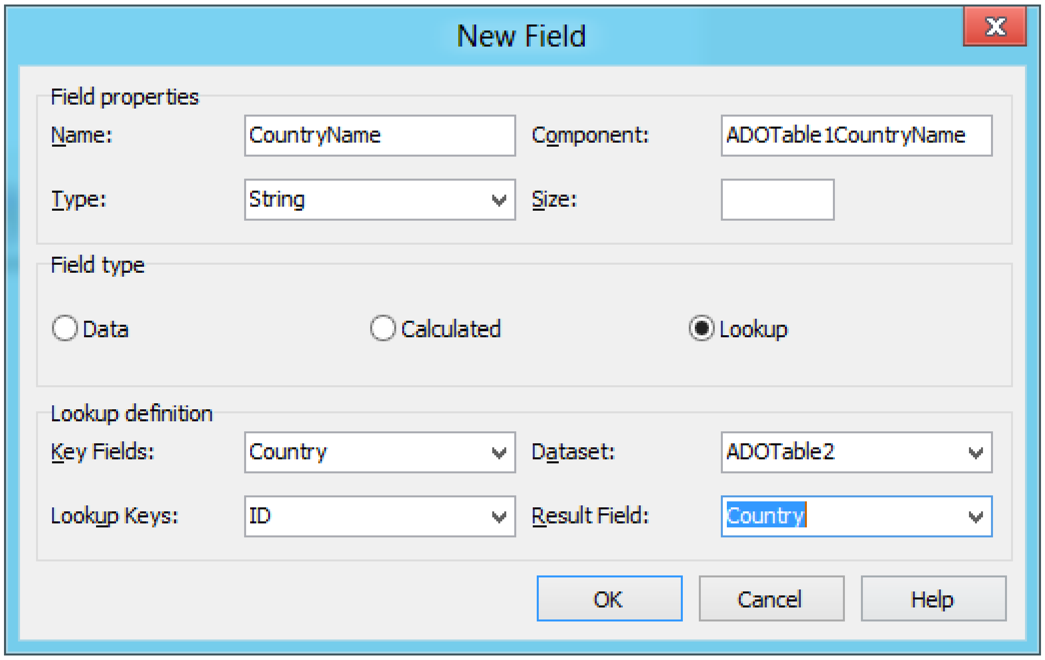
<u>Automatically</u>

The TTMSFMXGrid adds the capability of automatically detecting and add columns, along with a new visual designer and a livebindings wizard to help you create your application. More on this matter is explained in the LiveBindings Demo chapter.

The TTMSFMXGrid adds the capability of linking a column to a lookup combobox. Drop a TTMSFMXGrid on the form, or link it with a datasource of choice through the LiveBindings wizard. You will need a second datasource for the lookup table. In this case we have a field "Country" which is linked with a lookup table "Countries".



The CountryName field is based on the Country field and serves as a lookup field. Add a new field to the dataset and specify the lookup data:


New Field

Field properties

Name:	CountryName	Component:	ADOTable1CountryName
Type:	String	Size:	

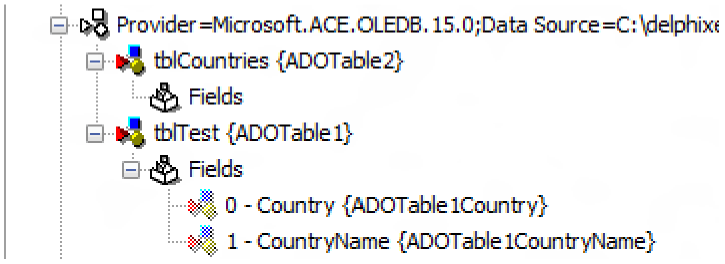
Field type

( ) Data ( ) Calculated [✓ Lookup]

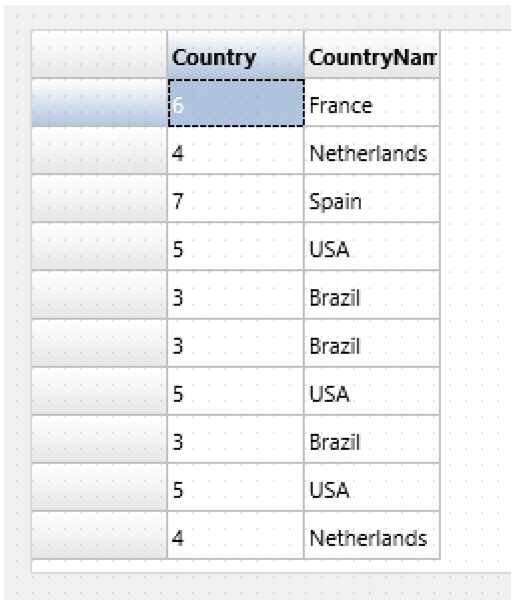
Lookup definition

Key Fields:	Country	Dataset:	ADOTable2
Lookup Keys:	ID	Result Field:	Country

You will need to add the Country field to the dataset in order to make the grid work properly:


```
Provider=Microsoft.ACE.OLEDB.15.0;Data Source=C:\delphix
  tblCountries {ADOTable2}
    Fields
  tblTest {ADOTable1}
    Fields
      0 - Country {ADOTable1Country}
      1 - CountryName {ADOTable1CountryName}
```

When starting the application, the grid will show 2 columns, one with the actual ID linked to the lookup table and the second column which is used to link the field to the lookup data.

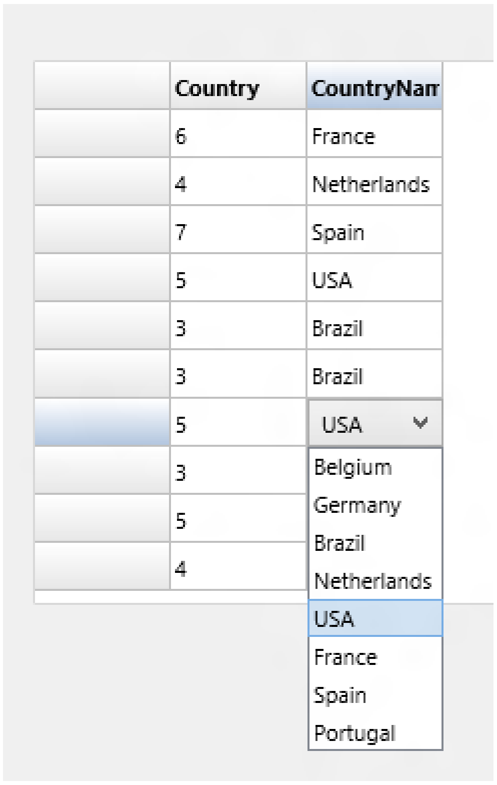


	Country	CountryNam
	6	France
	4	Netherlands
	7	Spain
	5	USA
	3	Brazil
	3	Brazil
	5	USA
	3	Brazil
	5	USA
	4	Netherlands

The grid automatically detects if a lookup field is used and uses a combobox for editing. The combobox is not automatically filled with the data from the lookup table. To link the data to the combobox insert the code below in the formcreate:

```
var
  LinkCountry: TLinkFillControlToField;
begin
  LinkCountry := TLinkFillControlToField.Create(Self);
  LinkCountry.Control := TMSFMXGrid1.CellComboBox;
  LinkCountry.DataSource := BindSourceDB1;
  LinkCountry.FieldName := 'Country';  // edit this field
  LinkCountry.FillDataSource := BindSourceDB2;
  LinkCountry.FillDisplayFieldName := 'Country';  // display this
field
  LinkCountry.FillValueFieldName := 'ID';  // data value for each item
  LinkCountry.Track := False; // Apply changes when click on item
rather than when exit control
  LinkCountry.Active := True;
```

This will bind the grid to the CellComboBox used inside the grid as an inplace editor. Double-clicking the cell starts editing and shows the combobox. The combobox is now filled with the values from the lookup table.



	Country	CountryNam
	6	France
	4	Netherlands
	7	Spain
	5	USA
	3	Brazil
	3	Brazil
	5	USA
	3	
	5	
	4	

A sample is included in the Distribution: LiveBindings Lookup demo.
The demo also shows how to link multiple lookup fields to a single editor combobox.

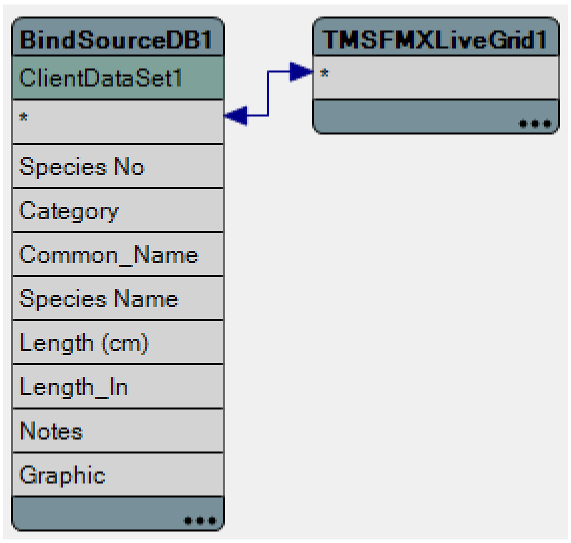
TTMSFMXGrid & TTMSFMXLiveGrid

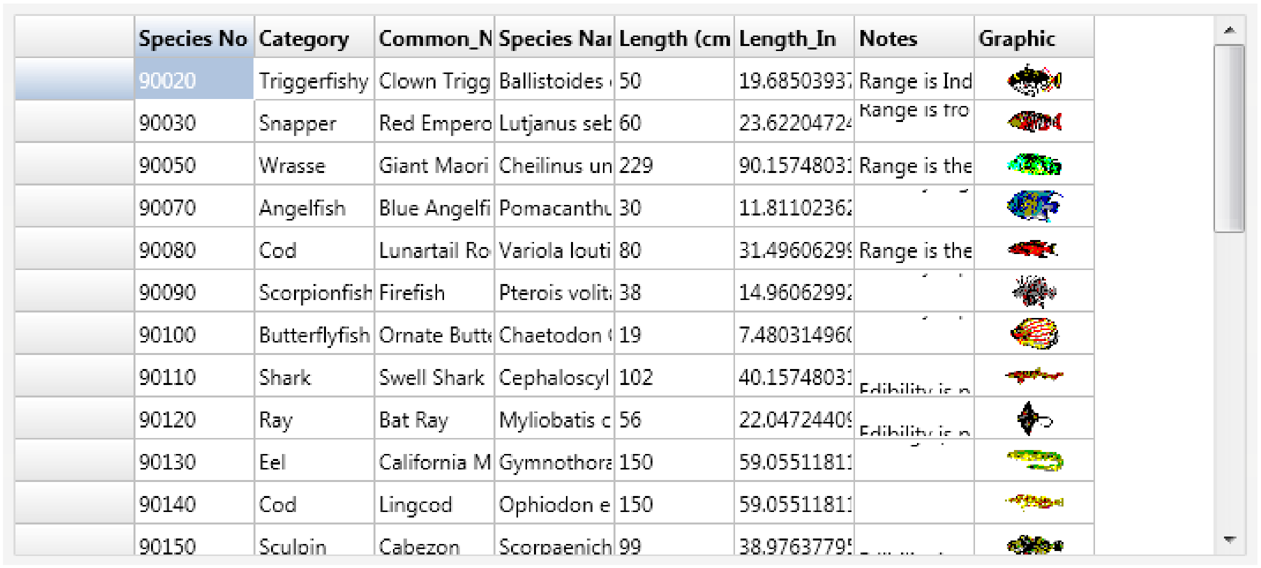
For binding data to the grid, there are 2 types of grids that can be used. The TTMSFMXGrid is able to bind to data and load all records at once. This type of grid can then optionally be disconnected by setting the property SaveDataSetData to true before deactivating the dataset to persist the data inside the grid.

After data is persisted and the grid doesn't have an active connection anymore, the grid can be used to apply filtering, grouping and sorting.

The TTMSFMXLiveGrid loads the data on demand. A database with many records loads faster than the TTMSFMXGrid implementation. The SaveDataSetData property doesn't have any effect on this grid because when the connection is deactivated the data is removed. Sorting, grouping and filtering on this grid directly are not supported. For each operation that has effect on the data, it must be applied directly on the dataset, which will automatically update the data inside the grid, if it has an active connection. Below is a sample how sorting is applied when the grid has an active connection.

Assuming you have already setup a connection to a dataset, either through manually implementing the columns and field connections or by connecting to a dataset and loading the columns/fields automatically the following code will sort the column on the dataset and automatically update the data. Note that sorting on the grid directly is not supported with an active connection so we will have to disable the built-in sorting. The sample below sorts a TClientDataSet loaded with data from the biolife.xml file.

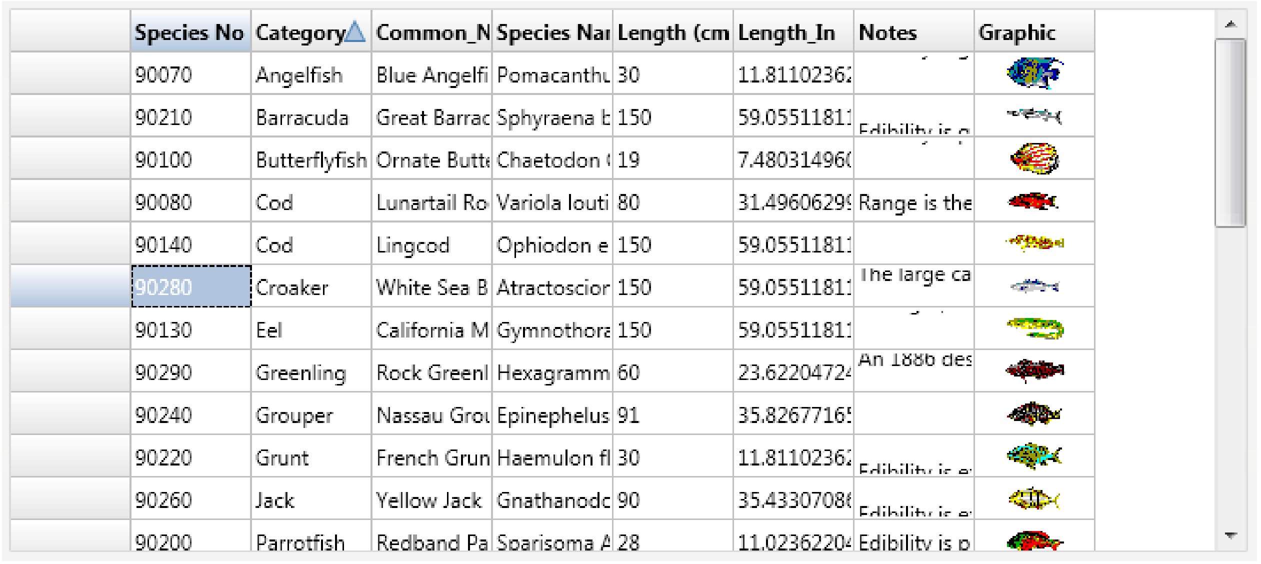




	Species No	Category	Common_N	Species Nar	Length (cm	Length_In	Notes	Graphic
	90020	Triggerfishy	Clown Trigg	Ballistoides	50	19.6850393	Range is Ind	
	90030	Snapper	Red Empero	Lutjanus seb	60	23.6220472	Range is fro	
	90050	Wrasse	Giant Maori	Cheilinus un	229	90.1574803	Range is the	
	90070	Angelfish	Blue Angelfi	Pomacanthu	30	11.8110236		
	90080	Cod	Lunartail Ro	Variola louti	80	31.4960629	Range is the	
	90090	Scorpionfish	Firefish	Pterois volit	38	14.9606299		
	90100	Butterflyfish	Ornate Butte	Chaetodon	19	7.48031496		
	90110	Shark	Swell Shark	Cephaloscyl	102	40.1574803	Edibility is n	
	90120	Ray	Bat Ray	Myliobatis c	56	22.0472440	Edibility is n	
	90130	Eel	California M	Gymnothora	150	59.0551181		
	90140	Cod	Lingcod	Ophiodon e	150	59.0551181		
	90150	Sculpin	Cabezon	Scorpaenich	99	38.9763779		

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  TMSFMXLiveGrid1.Options.Sorting.Mode := gsmNormal;
end;
```

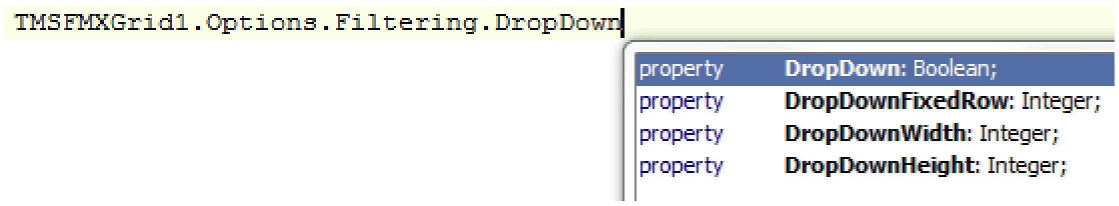
```
procedure TForm1.TMSFMXLiveGrid1CanSortColumn(Sender: TObject; ACol:
Integer;
  var Allow: Boolean);
var
  idx: TIndexDef;
  idxn: String;
  f: TField;
begin
  Allow := False; //disable built-in sorting
  f := ClientDataSet1.Fields[ACol - TMSFMXLiveGrid1.FixedColumns];
  if Assigned(f) then
  begin
    idxn := f.FullName + 'Index';
    if ClientDataSet1.IndexDefs.Count = 1 then
      idx := ClientDataset1.IndexDefs[0]
    else
      idx := ClientDataset1.IndexDefs.AddIndexDef;

    idx.Name := idxn;
    idx.Fields := f.FullName;
    ClientDataset1.IndexName := idxn;
    TMSFMXLiveGrid1.SortColumn := ACol;
  end;
end;
```



	Species No	Category	Common_N	Species Nai	Length (cm	Length_In	Notes	Graphic
	90070	Angelfish	Blue Angelfi	Pomacanthu	30	11.8110236		
	90210	Barracuda	Great Barrac	Sphyraena b	150	59.0551181	Edibility is	
	90100	Butterflyfish	Ornate Butt	Chaetodon	19	7.48031496		
	90080	Cod	Lunartail Ro	Variola louti	80	31.4960629	Range is the	
	90140	Cod	Lingcod	Ophiodon e	150	59.0551181		
	90280	Croaker	White Sea B	Atractoscior	150	59.0551181	The large ca	
	90130	Eel	California M	Gymnothora	150	59.0551181		
	90290	Greenling	Rock Greenl	Hexagramm	60	23.6220472	An 1886 des	
	90240	Grouper	Nassau Grou	Epinephelus	91	35.8267716		
	90220	Grunt	French Grun	Haemulon fl	30	11.8110236	Edibility is	
	90260	Jack	Yellow Jack	Gnathanodc	90	35.4330708	Edibility is	
	90200	Parrotfish	Redband Pa	Sparisoma A	28	11.0236220	Edibility is p	

Filtering

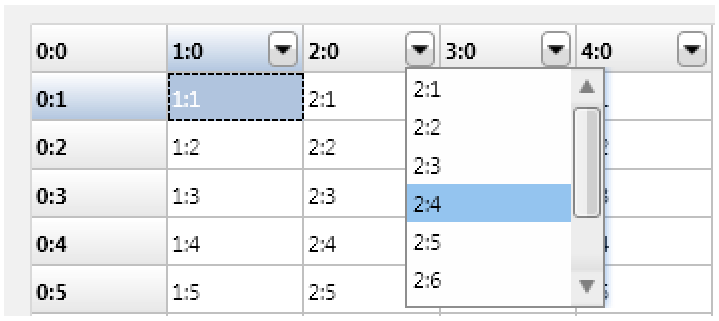
The TTMSFMXGrid also supports built-in filtering. Filtering can be done programmatically or via the user interface when enabling the dropdown button on a fixed column header cell.

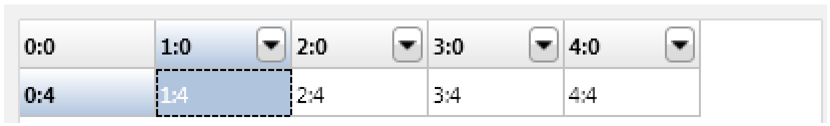


```
TMSFMXGrid1.Options.Filtering.DropDown
```

```
property    DropDown: Boolean;
property    DropDownFixedRow: Integer;
property    DropDownWidth: Integer;
property    DropDownHeight: Integer;
```

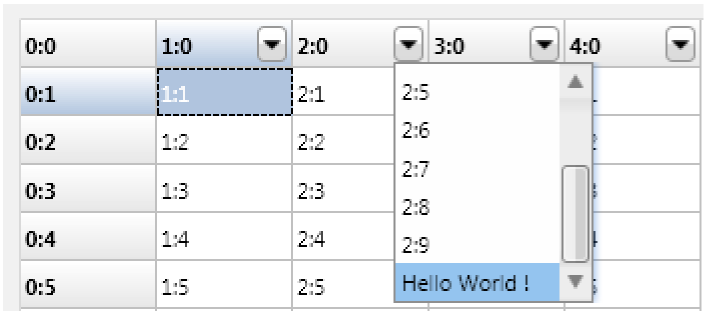
When the dropdown button is visible, the dropdown list is automatically filled with unique values from the column. When selecting an item from the dropdown list, the grid is filtered based on the value you have selected:





By default, when setting grid.Options.Filtering.DropDown = true, all normal column header cells get a dropdown button. With the OnNeedFilterDropDown, the dropdown button / filtering can be enabled / disabled per column. When a dropdown filter button is displayed, this dropdown list is automatically filled with the unique values in the column but the OnNeedFilterDropDownData event is triggered and this allows to alter the data that is displayed in the dropdown list.

```
procedure TForm1.TMSFMXGrid1NeedFilterDropDownData(Sender: TObject;
Col,
  Row: Integer; AValues: TStrings);
begin
  AValues.Add('Hello World !');
end;
```



When an item from the filter dropdown is selected, this triggers the OnFilterSelect event. This returns the column and the selected filter condition and also allows to dynamically change the condition.

As the filter dropdown is filled automatically with unique values of a column, it is by default not possible to undo a filter from the user-interface. With the help of the OnNeedFilterDropDownData and the OnFilterSelect event, an item can be added to the dropdown that will undo the filtering.

Add the (All) options to the dropdown:

```
procedure TForm1.TMSFMXGrid1NeedFilterDropDownData(Sender: TObject; Col,
  Row: Integer; AValues: TStrings);
begin
  AValues.Add('(All)');
end;
```

When the (All) option is selected, set the condition to accept all values:

```
procedure TForm1.TMSFMXGrid1FilterSelect(Sender: TObject; Col: Integer;
  var Condition: string);
begin
  if Condition = '(All)' then
  begin
    Condition := '*';
  end;
end;
```

Programmatically, a filter condition can be added to the filter list, and the list is filtered when applying the filter with grid.ApplyFilter;

```
with TMSFMXGrid1.Filter.Add do
begin
```

```
  Condition := '2:4';
  Column := 2;
end;
```

TMSFMXGrid.ApplyFilter;

To remove the filter again at a later time, call TMSFMXGrid.RemoveFilter;

The TFilterData type in the Filter collection has following properties:

Column: integer : sets the column the filter condition applies to
Condition: string : holds the condition, this is a string value including the use of <,>,&,|, *, ? specifiers
CaseSensitive: Boolean : defines whether the condition is case sensitive or not
Data: TFilterCells: specifies on what data the filter condition is applied. By default this is the cell text (fcNormal)
Prefix: string: part of the cell text that should be ignored (at start of the cell text)
Suffix: string: part of the cell text that should be ignored (at end of the cell text)
Operation: TFilterOperation :defines the logical operation between the filter condition and the previous filter condition.

The definition of TFilterOperation is:

foSHORT: short circuit Boolean evaluation
foNONE: no logical operation (typical for first filter condition)
foAND: logical AND
foXOR: logical XOR
foOR: logical OR

Example:

When a column contains numbers formatted like:

50.00USD
75.00USD
25.00USD
60.00USD

The filter to get values larger than 60, could be :

```
fd: TFilterData;
fd :=grid.Filter.Add;
fd.Column := 1;
fd.Suffix := ‘USD’
fd.Condition := ‘>50’;
```

To specify a filter that would retrieve values less than 30 or bigger than 60, this could be specified as:

```
fd: TFilterData;

fd :=grid.Filter.Add;
fd.Column := 1;
fd.Suffix := ‘USD’
fd.Condition := ‘>60’;

fd :=grid.Filter.Add;
fd.Column := 1;
fd.Suffix := ‘USD’
fd.Condition := ‘<30’;
fd.Operation := foOR;
```

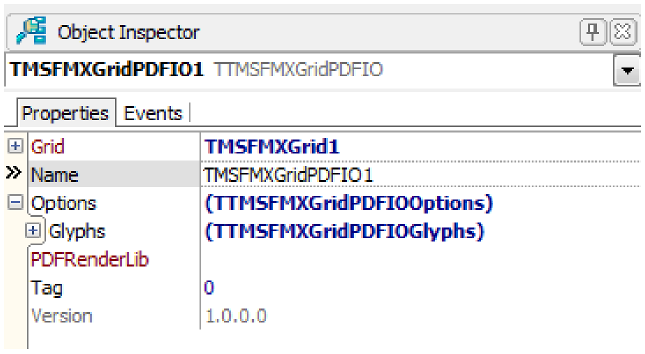
PDF Export

As explained in the ‘Printing’ chapter, the grid supports basic printing on an image and on a printer canvas by using the FMX.Printer unit, with various additional parameters to pass through the print procedure. This kind of ‘printing’ can be used to print the grid content on a pdf.

Unfortunately, printing on a pdf through this procedure doesn’t support text selection, and the quality is poor compared to a real pdf export engine. With the TTMSFMXGridPDFIO component you will be able to export the grid to a PDF on Windows, Mac and iOS.

Windows

The component has an implementation of the QuickPDF library that is used to export the grid on a pdf. When dropping an instance of the TTMSFMXGridPDFIO component on the form you will notice a PDFRenderLib and a Grid property.



Grid	TMSFMXGrid1
Name	TMSFMXGridPDFIO1
Options	(TTMSFMXGridPDFIOOptions)
Glyphs	(TTMSFMXGridPDFIOGlyphs)
PDFRenderLib	
Tag	0
Version	1.0.0.0

In Windows, the PDFRenderLib is nil by default and will throw an exception when trying to export. By adding the FMX.TMSQuickPDFRenderLib unit to the package, compiling and installing the package, this component is available and can be assigned to the PDFRenderLib property. The Quick PDF library (http://www.quickpdflibrary.com) can be downloaded as a 30-day trial version and contains zip files for various Delphi versions. Extract the correct version add the extracted zip file path to the library path in your IDE. After extraction, open the FMX.TMSQuickPDFRenderLib unit and paste your trial or registered license key inside the source file under

```
const AUnlockKey = 'Your trial / registered license key for QuickPDF';
```

The exporting is done by calling the ExportPDF procedure with a FileName parameter.

```
TMSFMXGridPDFIO1.ExportPDF('pdfexport.pdf');
```

The additional options such as a title, description, pagenumber and other appearance settings related to this PDF export component are set in the same way as the basic printing implementation on the grid. These options / properties can be found under grid.options.printing.

Mac / iOS

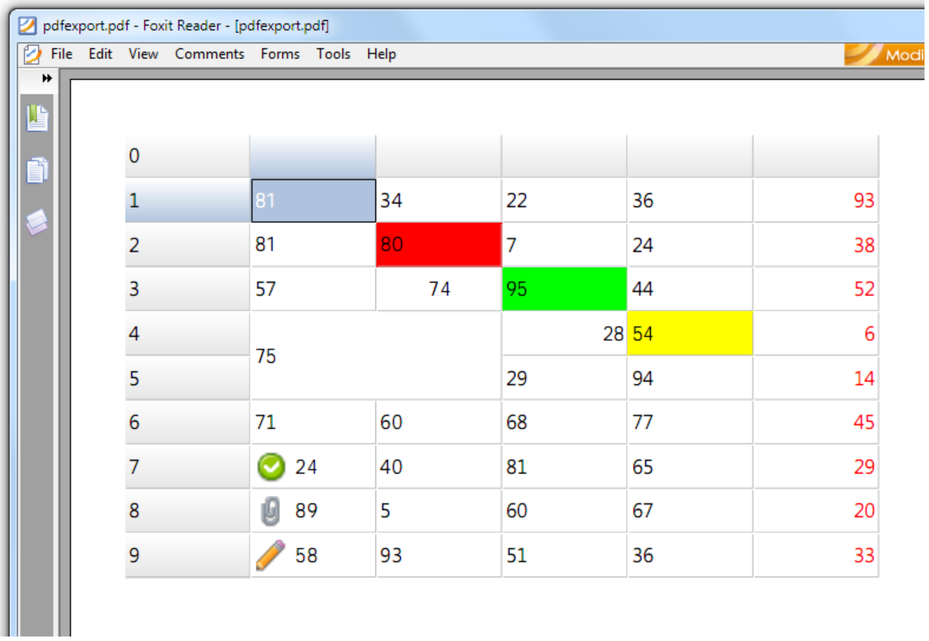
The Mac and iOS implementation doesn't require an additional PDFRenderLib instance and can be used directly by only connecting the grid and calling the ExportPDF procedure.
When building your application for iOS / Mac notice that the pdfexport.pdf file in the code snippet above, will not be generating a PDF in the same directory as the executable file as it is the case when exporting the pdf with the Windows application. For Mac or iOS, you can specify the root directory of the application or the documents directory of the Mac to output your PDF. In iOS, the Documents directory is a directory that is unique per application.

In the FMX.TMSXUtil unit, the XGetDocumentsDirectory and the XGetRootDirectory procedures can be used to export the pdf file to the correct directory.

```
TMSFMXGridPDFIO1.ExportPDF(XGetDocumentsDirectory + '/pdfexport.pdf');
```

Options

Checkboxes and radiobuttons are the only additional grid controls that can be exported to a PDF. Under options in the TTMSFMXGridPDFIO component you will notice that you can set an icon for these controls when exporting. By default, the icon is loaded from a resource file and can be overriden. Below is a result of an export to a PDF from a converted existing XLS export demo.



0					
1	81	34	22	36	93
2	81	80	7	24	38
3	57	74	95	44	52
4	75		28	54	6
5			29	94	14
6	71	60	68	77	45
7	✓ 24	40	81	65	29
8	📎 89	5	60	67	20
9	✏ 58	93	51	36	33

HTML formatted text, cell anchors, highlighting and marking in cells

The grid supports HTML formatted strings in cells. This is based on a small & fast HTML rendering engine. This engine implements a subset of the HTML standard to display formatted text. It supports following tags :

B : Bold tag
<B> : start bold text
</B> : end bold text

Example : This is a <B>test</B>

U : Underline tag
<U> : start underlined text
</U> : end underlined text

Example : This is a <U>test</U>

I : Italic tag
<I> : start italic text
</I> : end italic text

Example : This is a <I>test</I>

S : Strikeout tag
<S> : start strike-through text
</S> : end strike-through text

Example : This is a <S>test</S>

A : anchor tag
<A href="value"> : text after tag is an anchor. The 'value' after the href identifier is the anchor. This can be an URL (with ftp,http,mailto,file identifier) or any text.
If the value is an URL, the shellexecute function is called, otherwise, the anchor value can be found in the OnAnchorClick event </A> : end of anchor

Examples : This is a <A href= "mailto:myemail@mail.com ">test</A>
This is a <A href="http://www.tmssoftware.com">test</A>
This is a <A href="somevalue">test</A>

FONT : font specifier tag
<FONT face='facevalue' size='sizevalue' color='colorvalue' bgcolor='colorvalue'> : specifies font of text after tag.
with

- face : name of the font
- size : HTML style size if smaller than 5, otherwise pointsize of the font
- color : font color with either hexidecimal color specification or color constant name, ie claRed,claYellow,claWhite ... etc
- bgcolor : background color with either hexidecimal color specification or color constant name </FONT> : ends font setting

Examples : This is a <FONT face="Arial" size="12" color="clared">test</FONT>
This is a <FONT face="Arial" size="12" color="#FF0000">test</FONT>

P : paragraph
<P align="alignvalue" [bgcolor="colorvalue"] [bgcolorto="colorvalue"]> : starts a new paragraph, with left, right or center alignment. The paragraph background color is set by the optional bgcolor parameter. If bgcolor and bgcolorto are specified,
a gradient is displayed ranging from begin to end color.
</P> : end of paragraph

Example : <P align="right">This is a test</P>
Example : <P align="center">This is a test</P>
Example : <P align="left" bgcolor="#ff0000">This has a red background</P>
Example : <P align="right" bgcolor="claYellow">This has a yellow background</P>
Example : <P align="right" bgcolor="claYellow" bgcolorto="clared">This has a gradient background</P>*

HR : horizontal line
<HR> : inserts linebreak with horizontal line

BR : linebreak
<BR> : inserts a linebreak

BODY : body color / background specifier
<BODY bgcolor="colorvalue" [bgcolorto="colorvalue"] [dir="v|h"] background="imagefile specifier"> : sets the background color of the HTML text or the background bitmap file

Example : <BODY bgcolor="claYellow"> : sets background color to yellow
<BODY background="file://c:\test.bmp"> : sets tiled background to file test.bmp
<BODY bgcolor="claYellow" bgcolorto="claWhite" dir="v"> : sets a vertical gradient from yellow

to white

IND : indent tag
This is not part of the standard HTML tags but can be used to easily create multicolumn text
<IND x="indent"> : indents with "indent" pixels

Example :
This will be <IND x="75">indented 75 pixels.

IMG : image tag
<IMG src="specifier:name" [align="specifier"] [width="width"] [height="height"] [alt="specifier:name"] > : inserts an image at the location

specifier can be: name of image in a BitmapContainer

Optionally, an alignment tag can be included. If no alignment is included, the text alignment with respect to the image is bottom. Other possibilities are: align="top" and align="middle"

The width & height to render the image can be specified as well. If the image is embedded in anchor tags, a different image can be displayed when the mouse is in the image area through the Alt attribute.

Examples :
This is an image <IMG src="name">

SUB : subscript tag
<SUB> : start subscript text
</SUB> : end subscript text

Example : This is <SUP>9</SUP>/<SUB>16</SUB> looks like 9/16

SUP : superscript tag
<SUP> : start superscript text
</SUP> : end superscript text

UL : list tag
<UL> : start unordered list tag
</UL> : end unordered list

Example : <UL>

```
<LI>List item 1
<LI>List item 2
<UL>
<LI> Sub list item A
<LI> Sub list item B
</UL>
<LI>List item 3
</UL>
```

LI : list item
<LI [type="specifier"] [color="color"] [name="imagename"]>: new list item specifier can be "square", "circle" or "image" bullet. Color sets the color of the square or circle bullet. Imagename sets the PictureContainer image name for image to use as bullet

SHAD : text with shadow
<SHAD> : start text with shadow
</SHAD> : end text with shadow

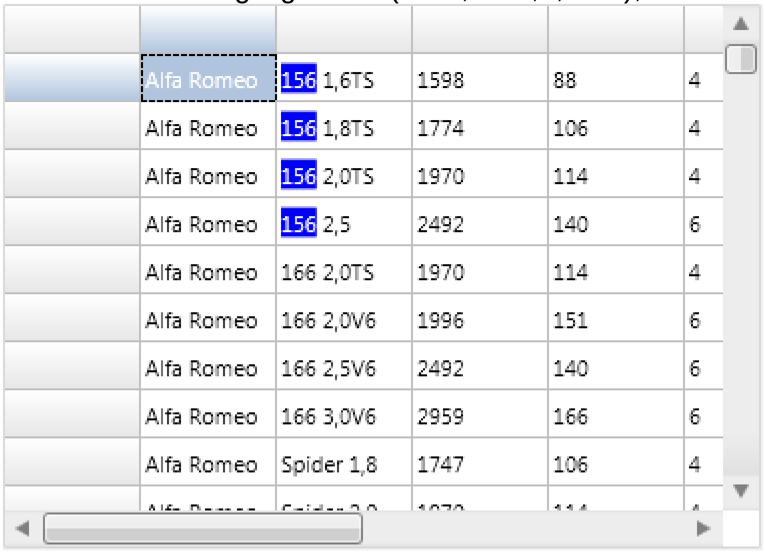
Z : hidden text
<Z> : start hidden text
</Z> : end hidden text

Special characters
Following standard HTML special characters are supported :
&lt; : less than : <
&gt; : greater than : >
&amp; : &
&quot; : "
&nbsp; : non breaking space
&trade; : trademark symbol
&euro; : euro symbol
&sect; : section symbol
&copy; : copyright symbol
&para; : paragraph symbol

When hyperlinks are specified in grid cells, these hyperlinks are displayed underlined and in blue color. When the hyperlink is clicked, the OnCellAnchorClick event is triggered. Via HTML formatting, the grid also offers highlighting or marking of text in grid cells. This can be used to indicate text that matches a search or to show errors. The following methods are available for marking & highlighting in cells:

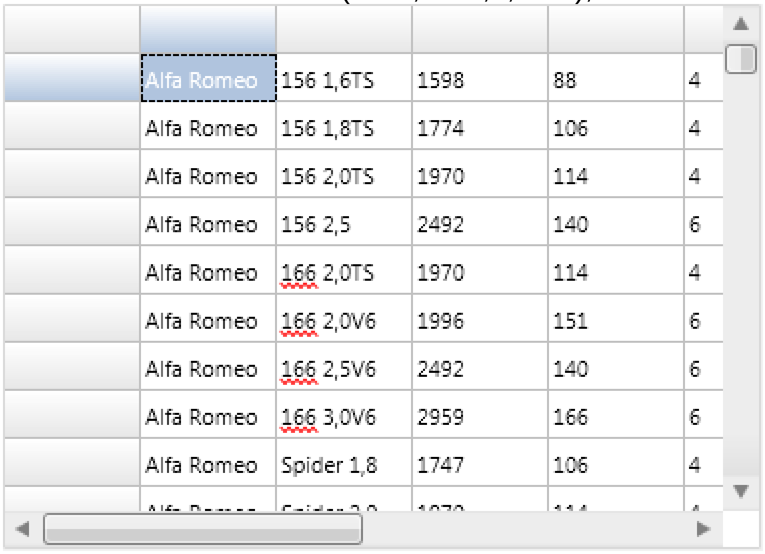
Examples:

TMSFMXGrid1.HighlightInCol(false,false,2,'156');



	Alfa Romeo	156 1,6TS	1598	88	4
	Alfa Romeo	156 1,8TS	1774	106	4
	Alfa Romeo	156 2,0TS	1970	114	4
	Alfa Romeo	156 2,5	2492	140	6
	Alfa Romeo	166 2,0TS	1970	114	4
	Alfa Romeo	166 2,0V6	1996	151	6
	Alfa Romeo	166 2,5V6	2492	140	6
	Alfa Romeo	166 3,0V6	2959	166	6
	Alfa Romeo	Spider 1,8	1747	106	4

TMSFMXGrid1.MarkInCol(false,false,2,'166');



	Alfa Romeo	156 1,6TS	1598	88	4
	Alfa Romeo	156 1,8TS	1774	106	4
	Alfa Romeo	156 2,0TS	1970	114	4
	Alfa Romeo	156 2,5	2492	140	6
	Alfa Romeo	166 2,0TS	1970	114	4
	Alfa Romeo	166 2,0V6	1996	151	6
	Alfa Romeo	166 2,5V6	2492	140	6
	Alfa Romeo	166 3,0V6	2959	166	6
	Alfa Romeo	Spider 1,8	1747	106	4

Available methods:

TMSFMXGrid.HighlightInCell(DoCase: Boolean; Col,Row: Integer; HiText: string);
Highlight the text HiText with or without case sensitivity in cell Col,Row.

TMSFMXGrid.HighlightInCol(DoFixed,DoCase: Boolean; Col: Integer; HiText: string);
Highlight the text HiText with or without case sensitivity in all cells in column Col.

TMSFMXGrid.HighlightInRow(DoFixed,DoCase: Boolean; Row: Integer; HiText: string);
Highlight the text HiText with or without case sensitivity in all cells in row Row.

TMSFMXGrid.HighlightInGrid(DoFixed,DoCase: Boolean; HiText: string);
Highlight the text HiText with or without case sensitivity in all cells in the grid.

TMSFMXGrid.UnHighlightInCell(Col,Row: Integer);
Remove the highlighting in cell Col,Row.

TMSFMXGrid.UnHighlightInCol(DoFixed: Boolean; Col: Integer);
Remove the highlighting in column Col, with or without fixed cells included.

TMSFMXGrid.UnHighlightInRow(DoFixed: Boolean; Row: Integer);
Remove the highlighting in row Row, with or without fixed cells included.

TMSFMXGrid.UnHighlightInGrid(DoFixed: Boolean);
Remove the highlighting in normal cells or all cells.

TMSFMXGrid.UnHighlightAll;
Remove the highlighting in all cells.

TMSFMXGrid.MarkInCell(DoCase: Boolean; Col,Row: Integer; HiText: string);
Mark the text HiText with or without case sensitivity in cell Col,Row.

TMSFMXGrid.MarkInCol(DoFixed,DoCase: Boolean; Col: Integer; HiText: string);
Mark the text HiText with or without case sensitivity in all cells in column Col.

TMSFMXGrid.MarkInRow(DoFixed,DoCase: Boolean; Row: Integer; HiText: string);
Mark the text HiText with or without case sensitivity in all cells in row Row.

TMSFMXGrid.MarkInGrid(DoFixed,DoCase: Boolean; HiText: string);
Mark the text HiText with or without case sensitivity in all cells in the grid.

TMSFMXGrid.UnMarkInCell(Col,Row: Integer);
Remove the marking in cell Col,Row.
TMSFMXGrid.UnMarkInCol(DoFixed: Boolean; Col: Integer);
Remove the marking in column Col, with or without fixed cells included.

TMSFMXGrid.UnMarkInRow(DoFixed: Boolean; Row: Integer);
Remove the marking in row Row, with or without fixed cells included.

TMSFMXGrid.UnMarkInGrid(DoFixed: Boolean);
Remove the marking in normal grid cells or all grid cells.

TMSFMXGrid.UnMarkAll;
Remove the marking in all grid cells.

General FireMonkey component usage guidlines

With the new FireMonkey framework, the methodology to create and use components has dramatically changed. A component now exists of 2 parts.

Visual part

The visual part is stored in a .style file, which is compiled to a .res file through an .rc file. The .rc file is included in the package and must be recompiled whenever a change is made to the .style file. For each component in this set you will find a .style file. In this file, the default layout of the component is stored.

You will notice different elements, basic elements such as an arc, ellipse, rectangle …
The elements combine and define the layout of a control. The basic elements are called shapes, and are already available by default. The TMS Instrumentation Workshop for FireMonkey takes this new way of styling one step further: custom shapes. In several components you will find custom shapes registered and useable in a new application, and used in the component by default.

Each shape or element can have a StyleName, which is used in the non-visual part of the control for interaction. This name is key in the relationship or "style-contract" between style resource and component code.

Non-visual part

The non-visual part of the component interacts with the shapes defined in the .style file. This is a normal .pas unit file as was used for VCL component, yet little to no painting is done in code. As explained above, the visual part is already defined by the style.

The component defined in this unit needs to inherit from the TStyledControl class, which can be styled at designtime. This is the base class for all styleable controls, just like the TCustomControl class was the base class for most controls in the VCL framework.

Naming convention

It is always good practice to handle a consistent naming convention, therefore all .rc, .pas files and .style files should start with the FireMonkey unit scope name "FMX.", such as the units: FMX.Types, FMX.Dialogs, FMX.Objects …

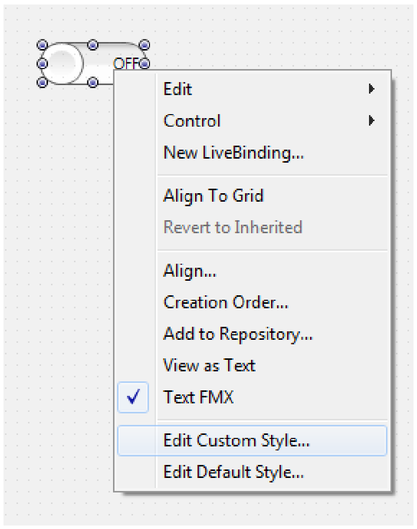
Inside the style file each element can have a StyleName, which can be used in the non-visual part to address the resource. Make sure each element has a unique StyleName to avoid mistakes when interacting with the component. All combinations of elements must be encapsulated within a rectangle element that is invisible by default (through the Fill.Kind and Stroke.Kind = bkNone), and has the StyleName of the component.

If you have a component named TFMXMyFirstControl, the the StyleName of the rectangle encapsulating all other elements must be set to FMXMyFirstControlStyle. The “T” is removed and “Style” is added.

Styling

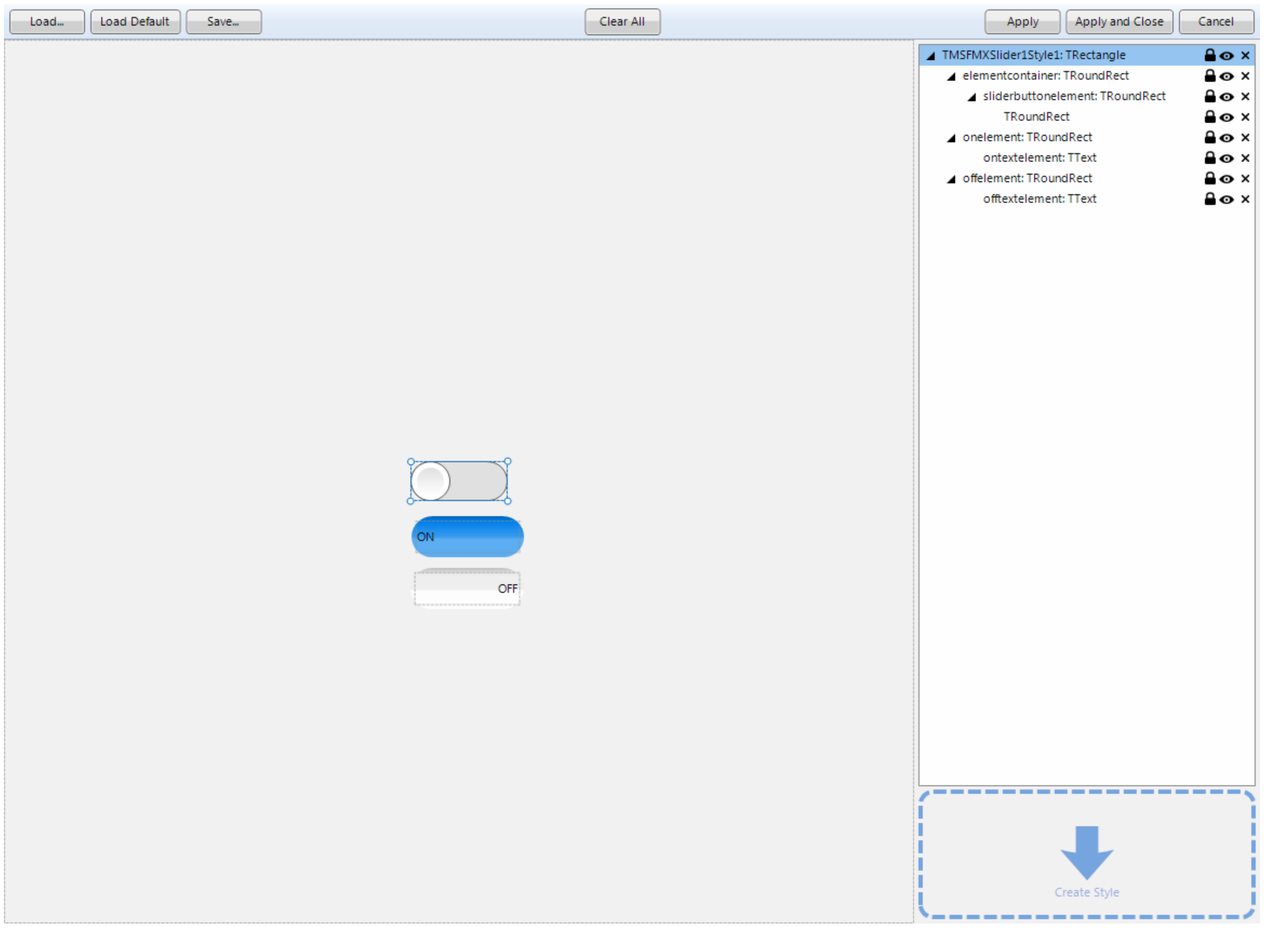
Each component inherits from TTMSFMXBaseControl which implements a basic Fill and Stroke, and handles the style resource files that define the default layout of the component. To change the visuals of the component you no longer have corresponding properties in the object inspector. Right-clicking on the component provides two extra menu items that can be used to edit the style of the component.

Clicking either of these items will automatically drop a StyleBook component on the form when there is not yet one available. A StyleBook holds custom and default styles. When the default style is changed, dropping a new component of the same class will automatically get this changed style as defined in the default style.



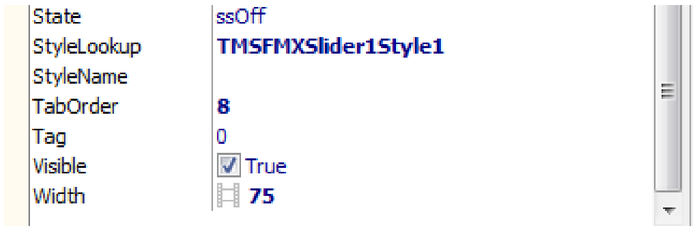
- Edit Custom Style: Clicking on this item starts the IDE style editor and copies the default style of the component. The name of the style is set to the component name on the form and appended with ‘Style1’. After changing properties through the editor, the style is then applied to the component. You will notice that the StyleLookUp property is set to the name of the custom style in the stylebook.

- Edit Default Style: Clicking on this item starts the IDE style editor and uses the default style of the component. As with the Edit Custom Style option, the name of that style is set. The difference between these 2 options is that the default style has a generic name and is applied to all new instances of the component that are dropped on the form. The StyleLoopup property is not set.



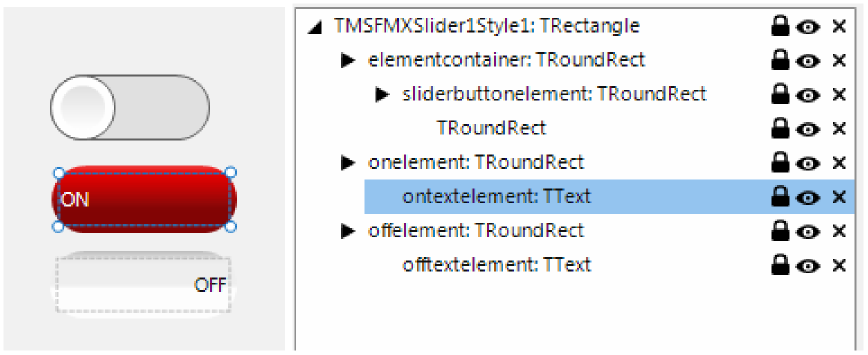
TMSFMXSlider1Style1: TRectangle
elementcontainer: TRoundRect
sliderbuttonelement: TRoundRect
TRoundRect
onelement: TRoundRect
ontextelement: TText
offelement: TRoundRect
offtextelement: TText

The IDE style editor can be started with these 2 options, or by double-clicking on the StyleBook editor icon on the form. In this example we have a TTMSFMXSlider component that will be altered with a custom style. Notice the TMSFMXSlider1Style1 name that is used for this style. When applying this style, you will also notice the StyleLookup property is set to TMSFMXSlider1Style1.



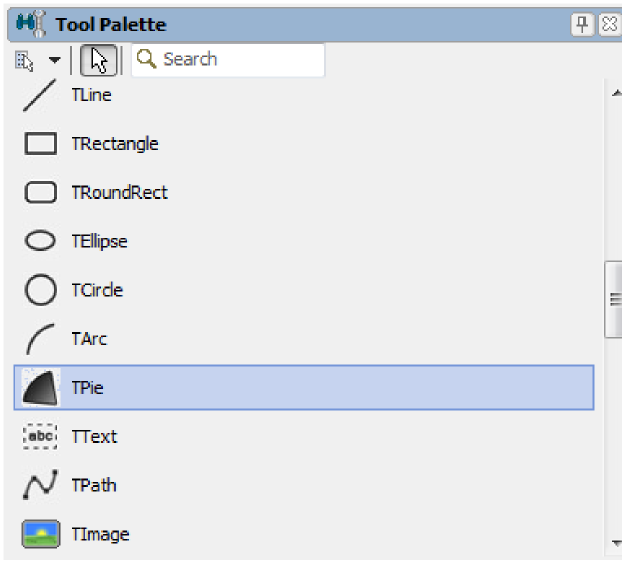
State	ssOff
StyleLookup	TMSFMXSlider1Style1
StyleName	
TabOrder	8
Tag	0
Visible	✓ True
Width	75

Each component exists of different styleable elements. Simple click on an element in the editor to change the appearance.

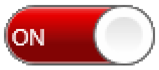


```
◢ TMSFMXSlider1Style1: TRectangle
  ▶ elementcontainer: TRoundRect
    ▶ sliderbuttonelement: TRoundRect
        TRoundRect
  ▶ onelement: TRoundRect
      ontextelement: TText
  ▶ offelement: TRoundRect
      offtextelement: TText
```

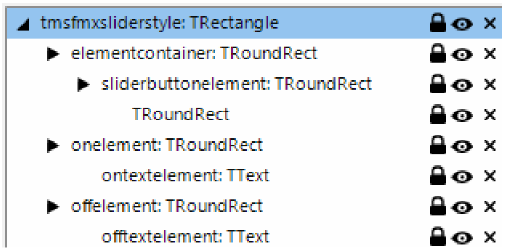
You can also add new elements from the Tool palette.



After applying the Style, the component will have the new custom style.



Dropping a new TTMSFMXSlider component on the form will not adopt this custom style and will have the default style applied. Editing the default style is done in the same way, yet the name of the style differs and each new instance of the TTMSFMXSlider adopts the edited default style.


tmsfmxsliderstyle: TRectangle
elementcontainer: TRoundRect
sliderbuttonelement: TRoundRect
TRoundRect
onelement: TRoundRect
ontextelement: TText
offelement: TRoundRect
offtextelement: TText

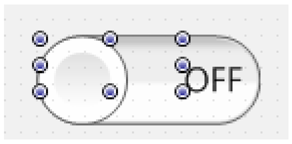
General component properties that do not directly define a visual appearance of the component are still displayed in the Object Inspector. Note though that some properties will affect what is available in the style editor! For example, if a component provides a collection of visible items displayed in the control and it is desirable that the visual appearance of each item can be customized, style elements (shapes) will be dynamically added or removed and be available in the IDE style editor.
In other cases, it is desirable that the appearance for a given type of items in a control is identical. This can be represented as a single style element in the style editor. The component will then internally copy the settings of the style element and apply it to each item displayed in the control.

Components

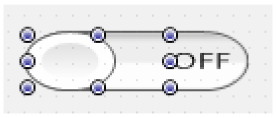
Most of the components in the FireMonkey framework can be scaled and rotated without loss of functionality and quality. As our base control implementation inherits from a base class which supports these features, all of the controls inside the TMS Instrumentation WorkShop set support scaling and rotation.

Scaling: With the Scale property you can specify how large the component must be. The default value of the X and Y property of the Scale is 1. This means that the default component layout is set at one, if you have a component which has 100 pixels width and height dimensions, setting the scale X and Y properties to 1.5 will automatically increase the width and height to 150 pixels. Below are some examples at designtime, which shows the capability of this property.

Scale 1.5

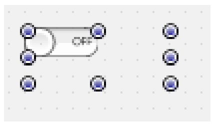


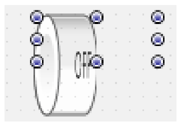
Scale X 1.5 Y 1



Scale 0.5

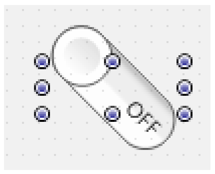
Scale X 0.5 Y 2





Rotation: The rotation property rotates the component around the center by default, which can be changed with the rotationcenter property. Rotating the component does not limit interaction capabilities and functionality.

45°

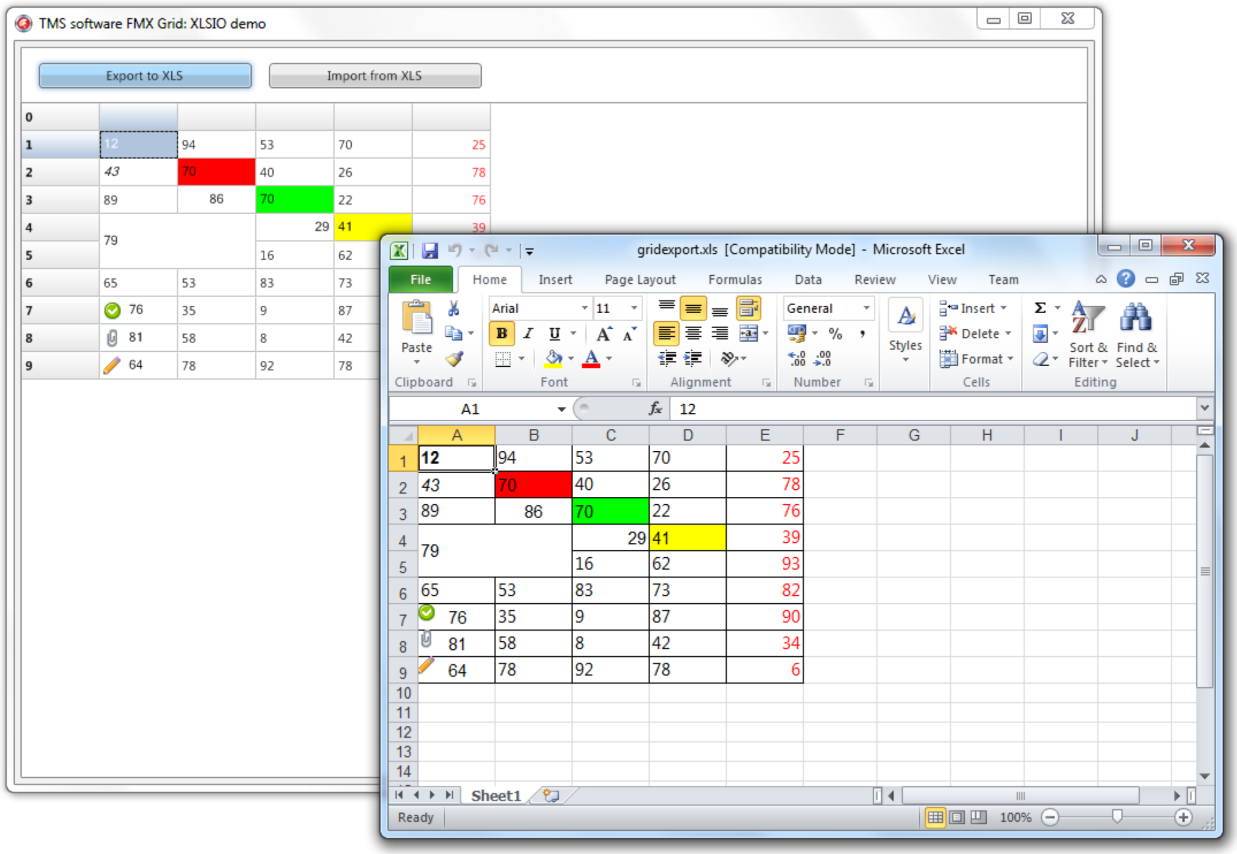


Samples

Included in the distribution is a set of ready to use applications that demonstrate the most important features of the grid. Below is an overview, screenshot and a short description of each demo.

1. XLSIODemo
2. GroupDemo
3. FilteringDemo
4. FixedFreezeDemo
5. ClipboardDemo
6. SortingDemo
7. CellControlsDemo
8. EditingDemo
9. MergingDemo
10. StylingDemo
11. RTFIODemo
12. EmbeddedControlsDemo
13. PrintingDemo
14. LiveBindings demo

XLSIODemo



0					
1	12	94	53	70	25
2	43	🟥 70	40	26	78
3	89	86	🟩 70	22	76
4	79		29	41	39
5			16	62	
6	65	53	83	73	
7	✓ 76	35	9	87	
8	81	58	8	42	
9	64	78	92	78	

A1 12

	A	B	C	D	E
1	12	94	53	70	25
2	43	🟥 70	40	26	78
3	89	86	🟩 70	22	76
4	79		29	41	39
5			16	62	93
6	65	53	83	73	82
7	✓ 76	35	9	87	90
8	81	58	8	42	34
9	64	78	92	78	6

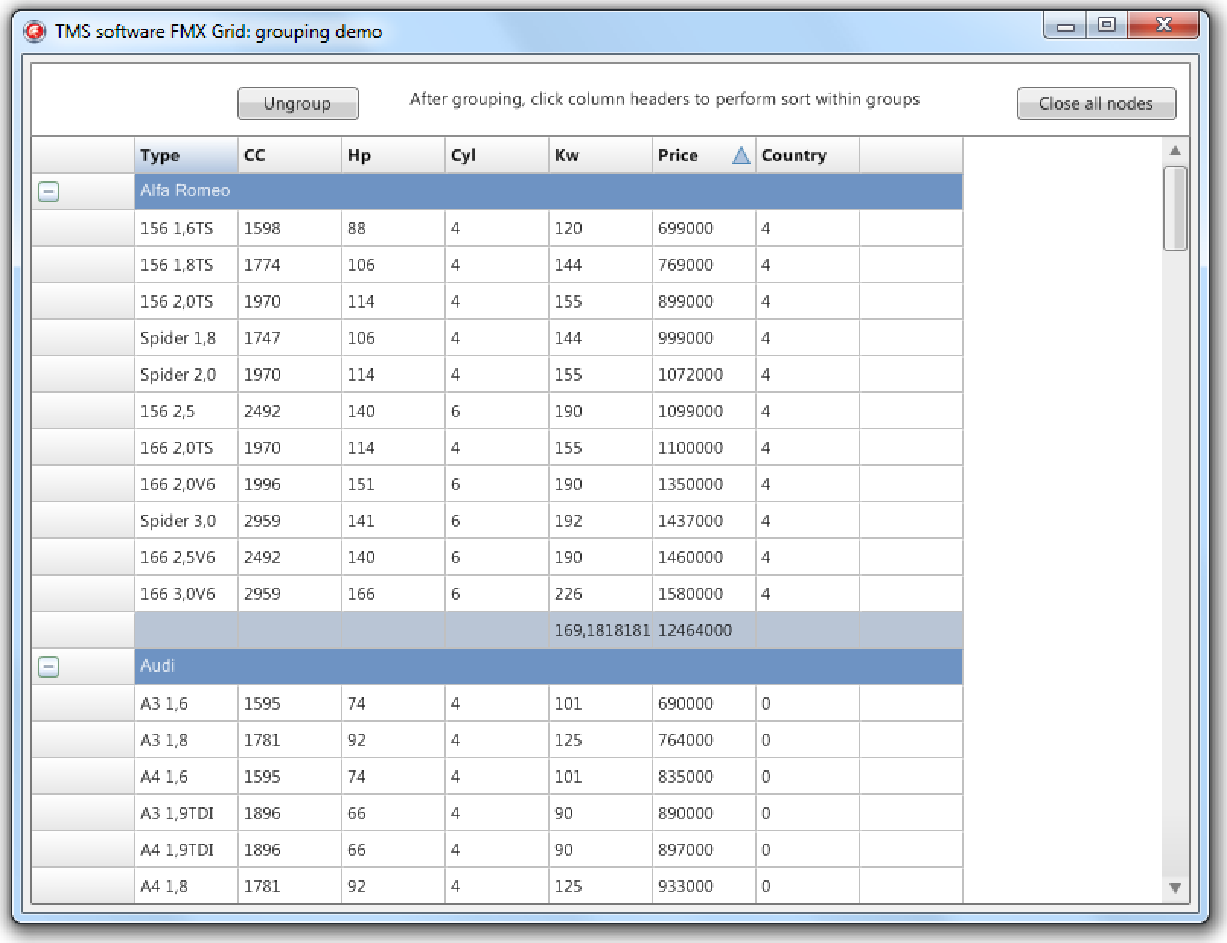
The XLS IO demo demonstrates exporting to and importing from an Excel compatible format. When clicking on the Export to XLS button, the grid is exported and the xls file is automatically opened if Excel is available.

Exporting the grid can be done with

```
TMSFMXGridExcelIO1.XLSExport('..\..\gridexport.xls');
XOpenFile('open', '..\..\gridexport.xls', '', '');
```

Note that for opening the file, the XOpenFile procedure is used that is located in the unit FMX.TMSXUtil which is compatible with all supported platforms which can handle file manipulation.

GroupDemo



TMS software FMX Grid: grouping demo

After grouping, click column headers to perform sort within groups

	Type	CC	Hp	Cyl	Kw	Price	Country	
⊟	Alfa Romeo							
	156 1,6TS	1598	88	4	120	699000	4	
	156 1,8TS	1774	106	4	144	769000	4	
	156 2,0TS	1970	114	4	155	899000	4	
	Spider 1,8	1747	106	4	144	999000	4	
	Spider 2,0	1970	114	4	155	1072000	4	
	156 2,5	2492	140	6	190	1099000	4	
	166 2,0TS	1970	114	4	155	1100000	4	
	166 2,0V6	1996	151	6	190	1350000	4	
	Spider 3,0	2959	141	6	192	1437000	4	
	166 2,5V6	2492	140	6	190	1460000	4	
	166 3,0V6	2959	166	6	226	1580000	4	
					169,1818181	12464000		
⊟	Audi							
	A3 1,6	1595	74	4	101	690000	0	
	A3 1,8	1781	92	4	125	764000	0	
	A4 1,6	1595	74	4	101	835000	0	
	A3 1,9TDI	1896	66	4	90	890000	0	
	A4 1,9TDI	1896	66	4	90	897000	0	
	A4 1,8	1781	92	4	125	933000	0	

Clicking the group button groups the Brand column and shows a group header and summary row with the total / average of the Kw and Price column. Clicking on a node collapses the group. To return to a normal state, click on the ungroup button. To close all nodes at once, click on the Close all nodes button.

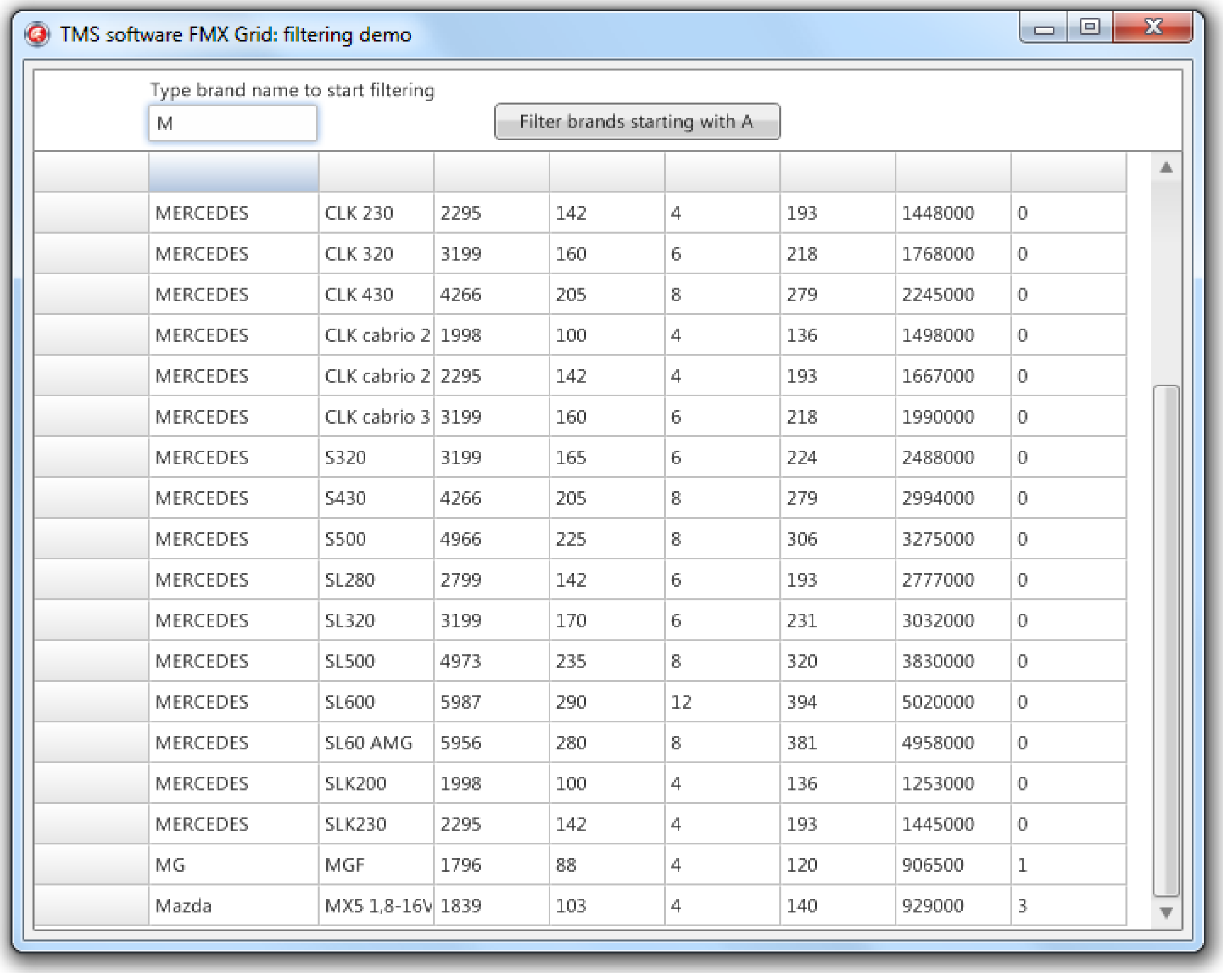
This demo's also demonstrates sorting in normal and grouped mode. Clicking on a column in group mode will sort the data specific to that group, resulting in a correct display of the sorted data.

To group grid data according to the demo use the following code below:

```
TMSFMXGrid1.SortData(1,sdAscending);
TMSFMXGrid1.Options.Grouping.MergeHeader := true;
```

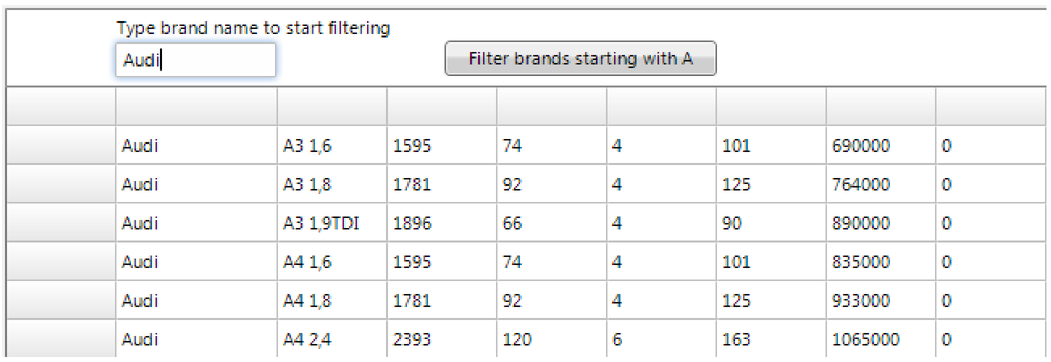
```
TMSFMXGrid1.Options.Grouping.Summary := true;

TMSFMXGrid1.Group(1);
TMSFMXGrid1.GroupSum(6);
TMSFMXGrid1.GroupAvg(5);
```

FilteringDemo


TMS software FMX Grid: filtering demo
Type brand name to start filtering
M
Filter brands starting with A

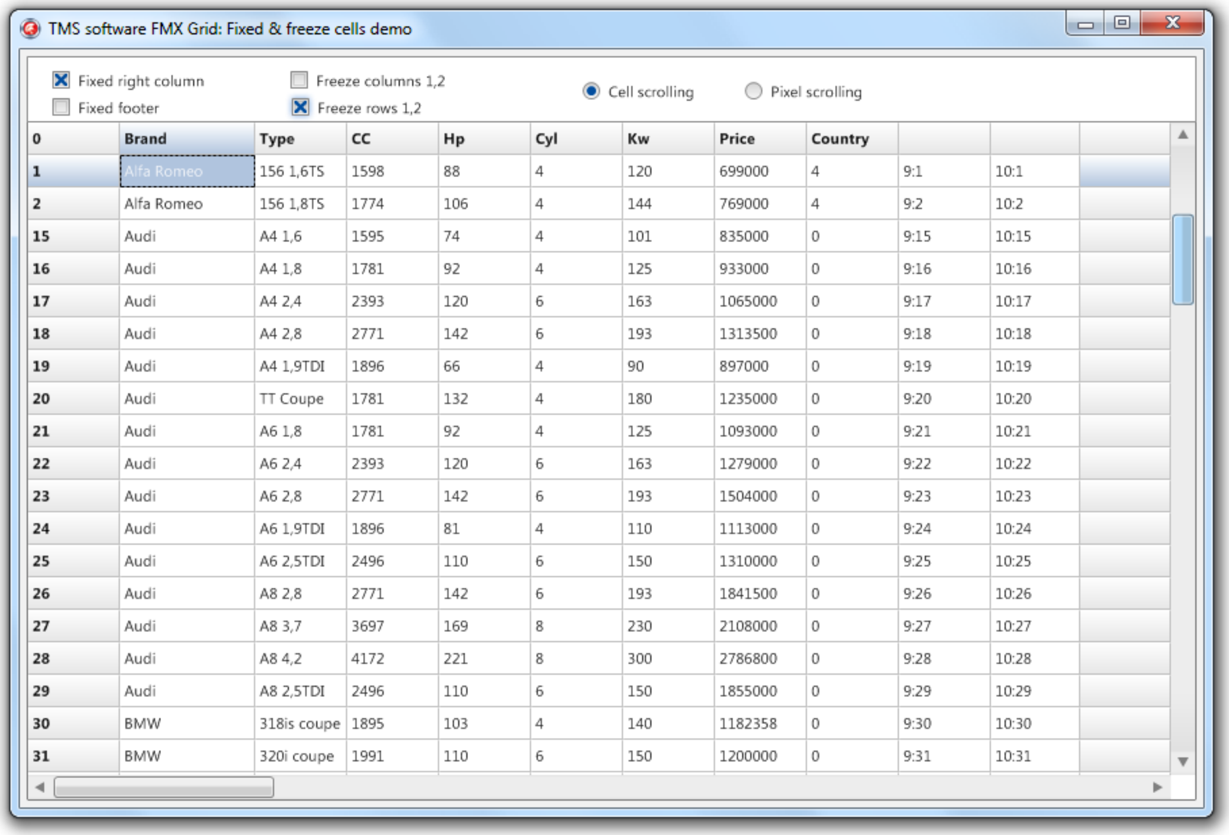
MERCEDES	CLK 230	2295	142	4	193	1448000	0	
MERCEDES	CLK 320	3199	160	6	218	1768000	0	
MERCEDES	CLK 430	4266	205	8	279	2245000	0	
MERCEDES	CLK cabrio 2	1998	100	4	136	1498000	0	
MERCEDES	CLK cabrio 2	2295	142	4	193	1667000	0	
MERCEDES	CLK cabrio 3	3199	160	6	218	1990000	0	
MERCEDES	S320	3199	165	6	224	2488000	0	
MERCEDES	S430	4266	205	8	279	2994000	0	
MERCEDES	S500	4966	225	8	306	3275000	0	
MERCEDES	SL280	2799	142	6	193	2777000	0	
MERCEDES	SL320	3199	170	6	231	3032000	0	
MERCEDES	SL500	4973	235	8	320	3830000	0	
MERCEDES	SL600	5987	290	12	394	5020000	0	
MERCEDES	SL60 AMG	5956	280	8	381	4958000	0	
MERCEDES	SLK200	1998	100	4	136	1253000	0	
MERCEDES	SLK230	2295	142	4	193	1445000	0	
MG	MGF	1796	88	4	120	906500	1	
Mazda	MX5 1,8-16V	1839	103	4	140	929000	3	

The filtering demo shows how the grid can be used to filter data, based on the first letter or a string typed in the edit box. Below is a result when typing a string in the edit box:


Type brand name to start filtering
Audi
Filter brands starting with A

Audi	A3 1,6	1595	74	4	101	690000	0
Audi	A3 1,8	1781	92	4	125	764000	0
Audi	A3 1,9TDI	1896	66	4	90	890000	0
Audi	A4 1,6	1595	74	4	101	835000	0
Audi	A4 1,8	1781	92	4	125	933000	0
Audi	A4 2,4	2393	120	6	163	1065000	0

FixedFreezeDemo



TMS software FMX Grid: Fixed & freeze cells demo

[✓] Fixed right column
[ ] Fixed footer
[ ] Freeze columns 1,2
[✓] Freeze rows 1,2
(●) Cell scrolling
( ) Pixel scrolling

0	Brand	Type	CC	Hp	Cyl	Kw	Price	Country			
1	Alfa Romeo	156 1,6TS	1598	88	4	120	699000	4	9:1	10:1	
2	Alfa Romeo	156 1,8TS	1774	106	4	144	769000	4	9:2	10:2	
15	Audi	A4 1,6	1595	74	4	101	835000	0	9:15	10:15	
16	Audi	A4 1,8	1781	92	4	125	933000	0	9:16	10:16	
17	Audi	A4 2,4	2393	120	6	163	1065000	0	9:17	10:17	
18	Audi	A4 2,8	2771	142	6	193	1313500	0	9:18	10:18	
19	Audi	A4 1,9TDI	1896	66	4	90	897000	0	9:19	10:19	
20	Audi	TT Coupe	1781	132	4	180	1235000	0	9:20	10:20	
21	Audi	A6 1,8	1781	92	4	125	1093000	0	9:21	10:21	
22	Audi	A6 2,4	2393	120	6	163	1279000	0	9:22	10:22	
23	Audi	A6 2,8	2771	142	6	193	1504000	0	9:23	10:23	
24	Audi	A6 1,9TDI	1896	81	4	110	1113000	0	9:24	10:24	
25	Audi	A6 2,5TDI	2496	110	6	150	1310000	0	9:25	10:25	
26	Audi	A8 2,8	2771	142	6	193	1841500	0	9:26	10:26	
27	Audi	A8 3,7	3697	169	8	230	2108000	0	9:27	10:27	
28	Audi	A8 4,2	4172	221	8	300	2786800	0	9:28	10:28	
29	Audi	A8 2,5TDI	2496	110	6	150	1855000	0	9:29	10:29	
30	BMW	318is coupe	1895	103	4	140	1182358	0	9:30	10:30	
31	BMW	320i coupe	1991	110	6	150	1200000	0	9:31	10:31	

This demo demonstrates the use of fixed and freeze columns / rows, which are cells that do not scroll along when navigating through the grid.

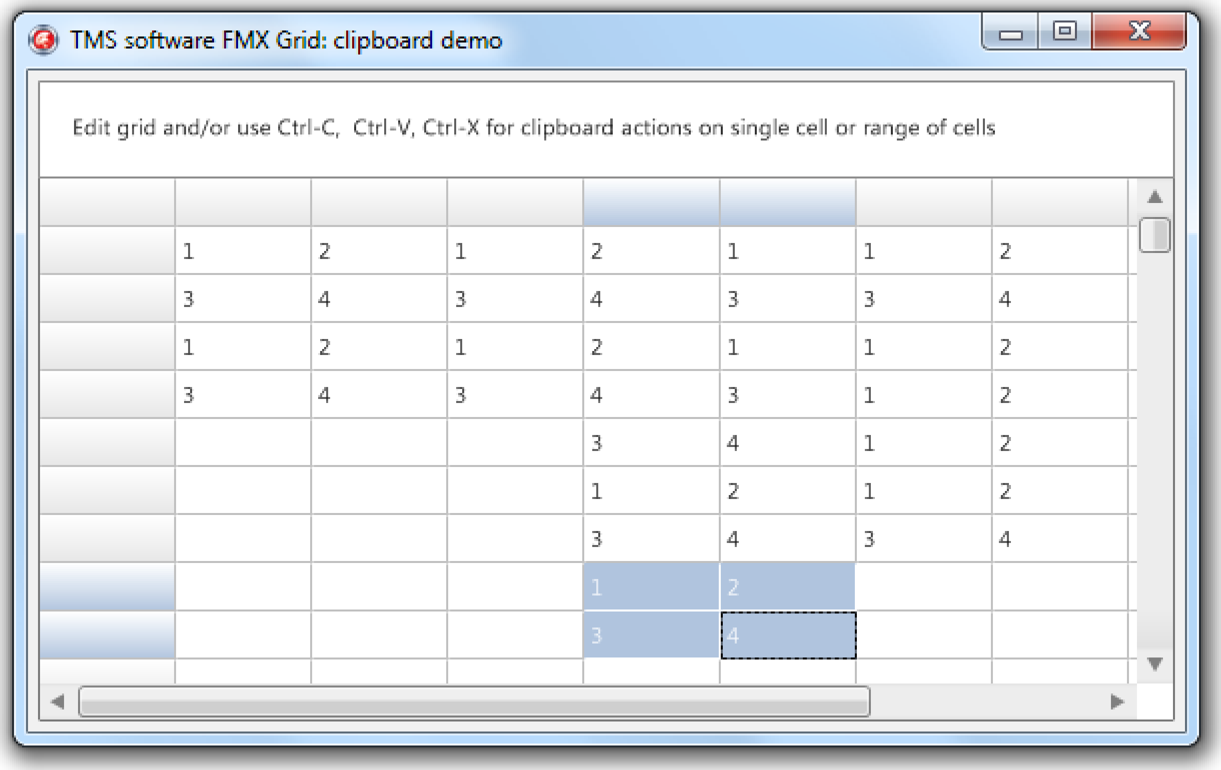
The fixed columns / rows are always displayed starting from the first column/row and are drawn in a fixed layout. These cells cannot be selected and are only used, amongst other features, for sorting, filtering and grouping interaction capabilities.

The Freeze columns / rows are displayed starting after the fixed columns /rows are displayed and are drawn in a normal layout. The cells also act as normal cells. The only difference is that they do not scroll when navigating through the grid.

The Fixed Right Columns / Fixed Footer rows are additional fixed cells that are displayed respectively at the right side and bottom side of the grid and do not scroll when navigating through the grid.

The last option is switching between cell scrolling and pixel scrolling.

ClipboardDemo



TMS software FMX Grid: clipboard demo

Edit grid and/or use Ctrl-C, Ctrl-V, Ctrl-X for clipboard actions on single cell or range of cells

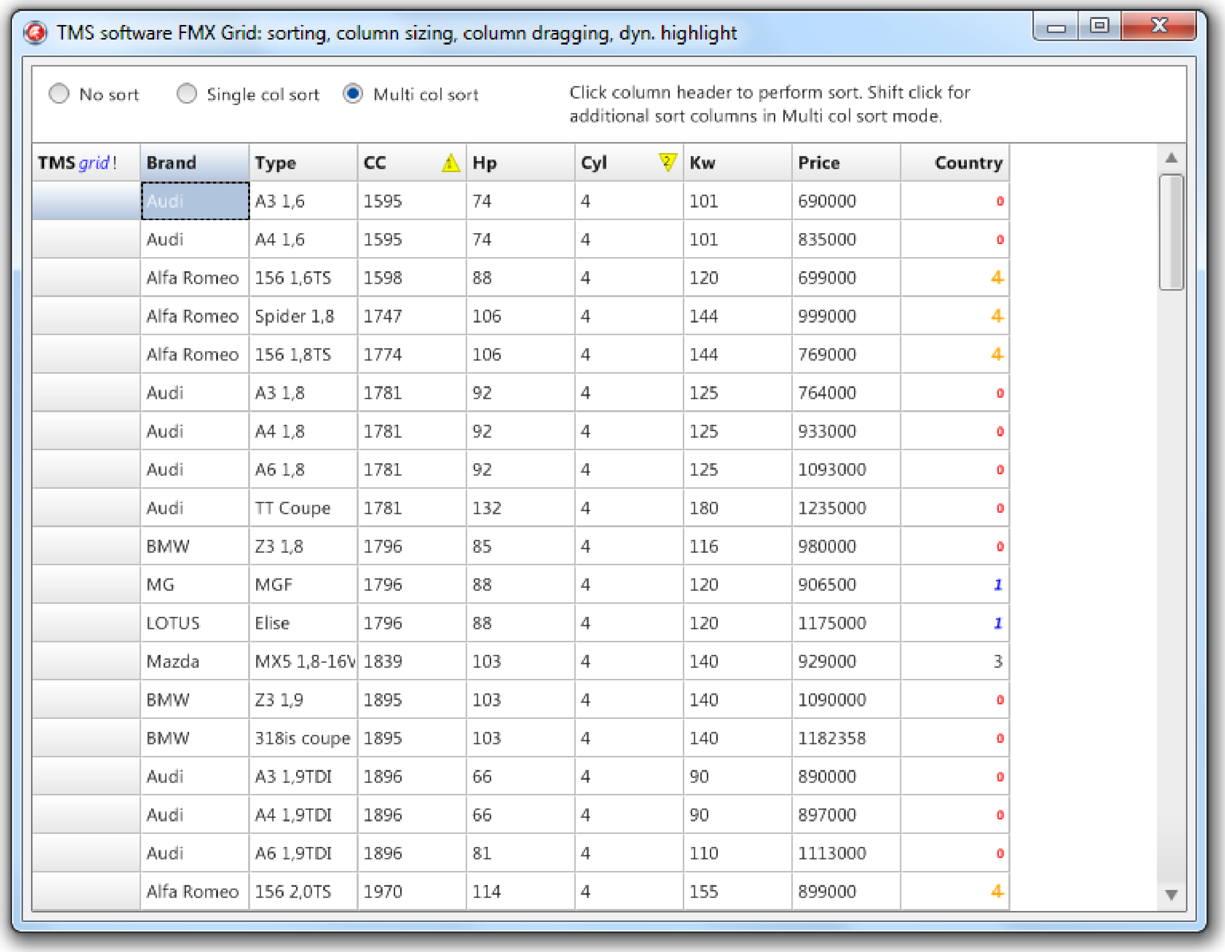
	1	2	1	2	1	1	2
	3	4	3	4	3	3	4
	1	2	1	2	1	1	2
	3	4	3	4	3	1	2
				3	4	1	2
				1	2	1	2
				3	4	3	4
				1	2		
				3	4		

This demo demonstrates clipboard support. The Cut, Copy and Paste actions are integrated in the grid with the common known keyboard shortcuts. There are additional properties that can be set when pasting the data, such as disabling overwriting readonly cells, automatically appending columns and rows when necessary. All these options can be found under grid.options.Clipboard.

These clipboard actions can also be done programmatically.

```
TMSFMXGrid1.CutToClipboard;
TMSFMXGrid1.PasteFromClipboard;
TMSFMXGrid1.CopyToClipboard;
```

SortingDemo



TMS software FMX Grid: sorting, column sizing, column dragging, dyn. highlight

No sort · Single col sort · [✓ Multi col sort]

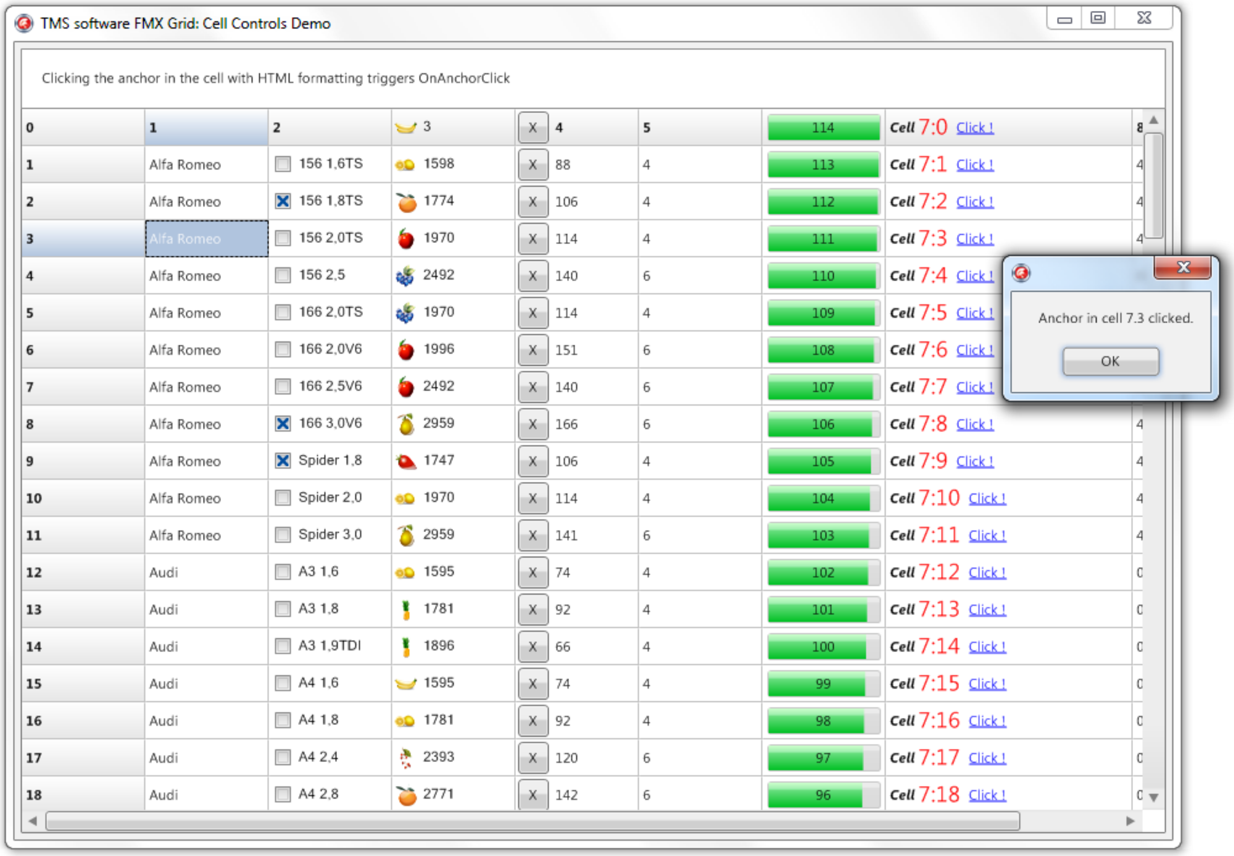
Click column header to perform sort. Shift click for additional sort columns in Multi col sort mode.

TMS grid!	Brand	Type	CC	Hp	Cyl	Kw	Price	Country
	Audi	A3 1,6	1595	74	4	101	690000	0
	Audi	A4 1,6	1595	74	4	101	835000	0
	Alfa Romeo	156 1,6TS	1598	88	4	120	699000	4
	Alfa Romeo	Spider 1,8	1747	106	4	144	999000	4
	Alfa Romeo	156 1,8TS	1774	106	4	144	769000	4
	Audi	A3 1,8	1781	92	4	125	764000	0
	Audi	A4 1,8	1781	92	4	125	933000	0
	Audi	A6 1,8	1781	92	4	125	1093000	0
	Audi	TT Coupe	1781	132	4	180	1235000	0
	BMW	Z3 1,8	1796	85	4	116	980000	0
	MG	MGF	1796	88	4	120	906500	1
	LOTUS	Elise	1796	88	4	120	1175000	1
	Mazda	MX5 1,8-16V	1839	103	4	140	929000	3
	BMW	Z3 1,9	1895	103	4	140	1090000	0
	BMW	318is coupe	1895	103	4	140	1182358	0
	Audi	A3 1,9TDI	1896	66	4	90	890000	0
	Audi	A4 1,9TDI	1896	66	4	90	897000	0
	Audi	A6 1,9TDI	1896	81	4	110	1113000	0
	Alfa Romeo	156 2,0TS	1970	114	4	155	899000	4

The grid supports 2 sorting modes demonstrated in this demo. The first mode is the normal mode which only allows sorting on one column and is indicated with a blue triangle displaying the column that is sorted and displaying the sort direction.

When switching the multi column sorting, the blue triangles change to yellow triangles displaying the index number (default 1). In this mode multiple columns can be selected and sorted with the shift key. When releasing the key and clicking on a column, the grid returns to sorting on 1 column.

CellControlsDemo



TMS software FMX Grid: Cell Controls Demo

Clicking the anchor in the cell with HTML formatting triggers OnAnchorClick

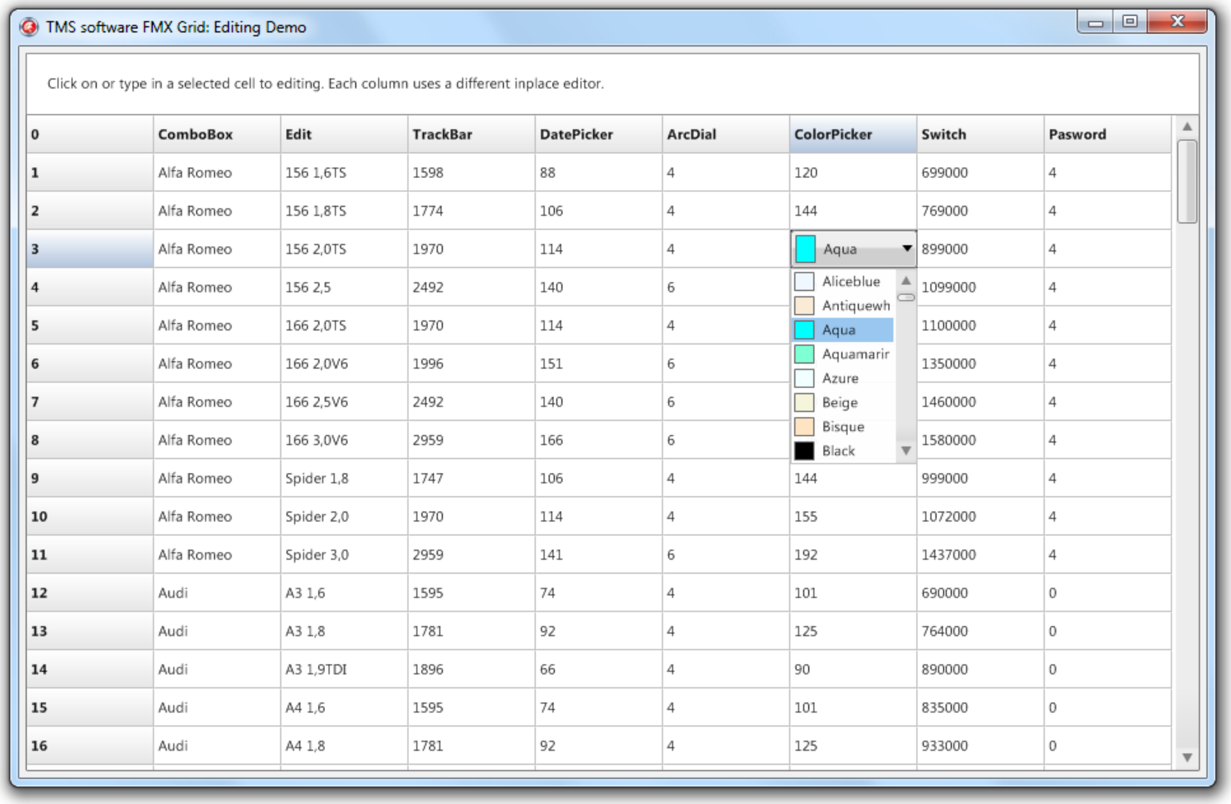
0	1	2	3	X 4	5	114	Cell 7:0 Click !	8
1	Alfa Romeo	☐ 156 1,6TS	1598	X 88	4	113	Cell 7:1 Click !	4
2	Alfa Romeo	☒ 156 1,8TS	1774	X 106	4	112	Cell 7:2 Click !	4
3	Alfa Romeo	☐ 156 2,0TS	1970	X 114	4	111	Cell 7:3 Click !	4
4	Alfa Romeo	☐ 156 2,5	2492	X 140	6	110	Cell 7:4 Click !	
5	Alfa Romeo	☐ 166 2,0TS	1970	X 114	4	109	Cell 7:5 Click !	
6	Alfa Romeo	☐ 166 2,0V6	1996	X 151	6	108	Cell 7:6 Click !	
7	Alfa Romeo	☐ 166 2,5V6	2492	X 140	6	107	Cell 7:7 Click !	
8	Alfa Romeo	☒ 166 3,0V6	2959	X 166	6	106	Cell 7:8 Click !	4
9	Alfa Romeo	☒ Spider 1,8	1747	X 106	4	105	Cell 7:9 Click !	4
10	Alfa Romeo	☐ Spider 2,0	1970	X 114	4	104	Cell 7:10 Click !	4
11	Alfa Romeo	☐ Spider 3,0	2959	X 141	6	103	Cell 7:11 Click !	4
12	Audi	☐ A3 1,6	1595	X 74	4	102	Cell 7:12 Click !	0
13	Audi	☐ A3 1,8	1781	X 92	4	101	Cell 7:13 Click !	0
14	Audi	☐ A3 1,9TDI	1896	X 66	4	100	Cell 7:14 Click !	0
15	Audi	☐ A4 1,6	1595	X 74	4	99	Cell 7:15 Click !	0
16	Audi	☐ A4 1,8	1781	X 92	4	98	Cell 7:16 Click !	0
17	Audi	☐ A4 2,4	2393	X 120	6	97	Cell 7:17 Click !	0
18	Audi	☐ A4 2,8	2771	X 142	6	96	Cell 7:18 Click !	0

Anchor in cell 7.3 clicked.

This demo demonstrates the use of cell controls, which are different from standard cells. Each cell in the grid is an object that can contain other objects. This is designed in such way that it fully supports the FireMonkey design philosophy.

The grid already implements a set of base control cells such as a checkbox, radiobutton and bitmap grid cell. This demo also shows how to add custom controls and how interact with them.

EditingDemo



Click on or type in a selected cell to editing. Each column uses a different inplace editor.

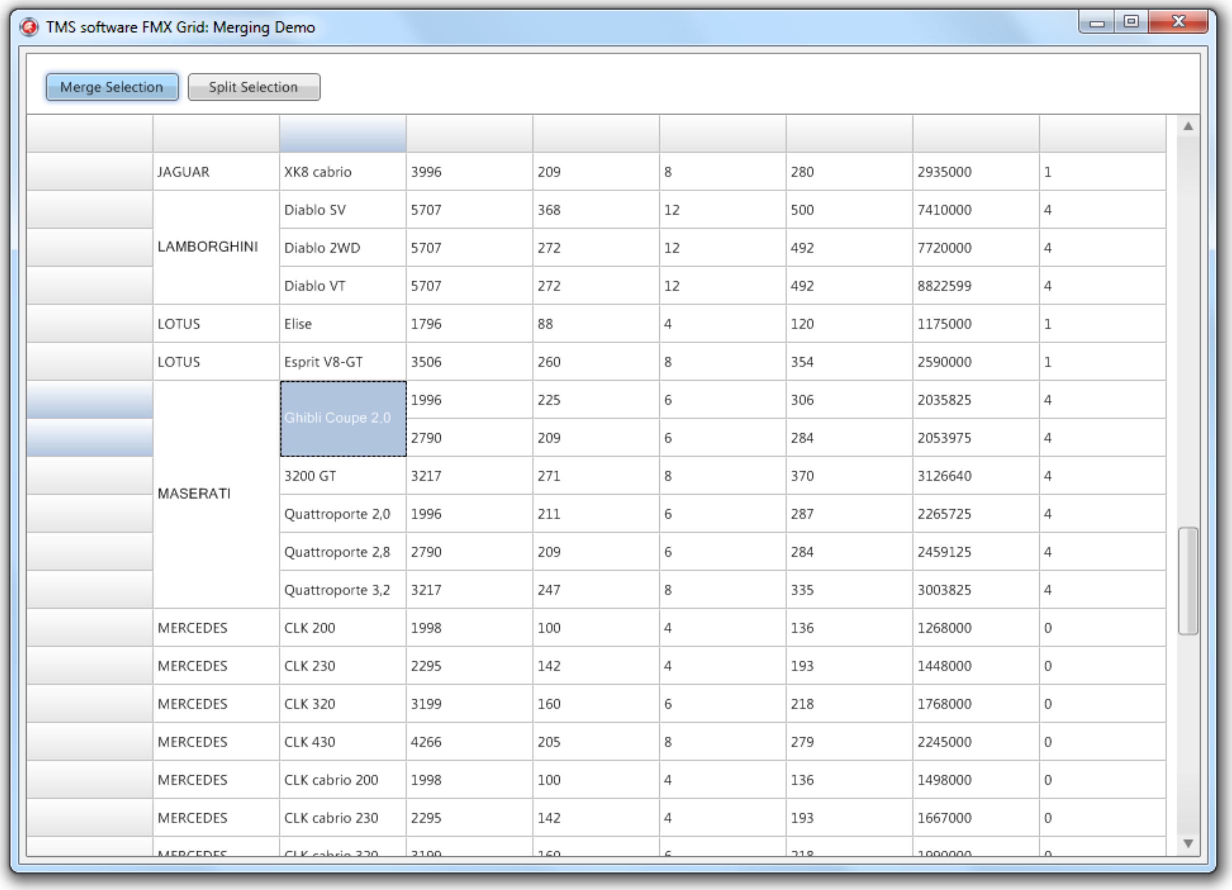
0	ComboBox	Edit	TrackBar	DatePicker	ArcDial	ColorPicker	Switch	Pasword
1	Alfa Romeo	156 1,6TS	1598	88	4	120	699000	4
2	Alfa Romeo	156 1,8TS	1774	106	4	144	769000	4
3	Alfa Romeo	156 2,0TS	1970	114	4	Aqua	899000	4
4	Alfa Romeo	156 2,5	2492	140	6		1099000	4
5	Alfa Romeo	166 2,0TS	1970	114	4		1100000	4
6	Alfa Romeo	166 2,0V6	1996	151	6		1350000	4
7	Alfa Romeo	166 2,5V6	2492	140	6		1460000	4
8	Alfa Romeo	166 3,0V6	2959	166	6		1580000	4
9	Alfa Romeo	Spider 1,8	1747	106	4	144	999000	4
10	Alfa Romeo	Spider 2,0	1970	114	4	155	1072000	4
11	Alfa Romeo	Spider 3,0	2959	141	6	192	1437000	4
12	Audi	A3 1,6	1595	74	4	101	690000	0
13	Audi	A3 1,8	1781	92	4	125	764000	0
14	Audi	A3 1,9TDI	1896	66	4	90	890000	0
15	Audi	A4 1,6	1595	74	4	101	835000	0
16	Audi	A4 1,8	1781	92	4	125	933000	0

The grid supports editing that is enabled by default. In this demo, various supported editor types are demonstrated as well as setting additional properties through various helper events that are triggered when editing.

By default the editor type is an edit that supports lookup and autocompletion, as well as validation with several edit types. To change the editor type the demo implements the OnGetCellEditorType event.

```
procedure TForm1.TMSFMXGrid1GetCellEditorType(Sender: TObject; ACol,
  ARow: Integer; var CellEditorType: TTMSFMXGridEditorType);
begin
  case ACol of
    1: CellEditorType := etComboBox;
    3: CellEditorType := etTrackBar;
    4: CellEditorType := etDatePicker;
    5: CellEditorType := etArcDial;
    6: CellEditorType := etColorComboBox;
  end;
end;
```

MergingDemo



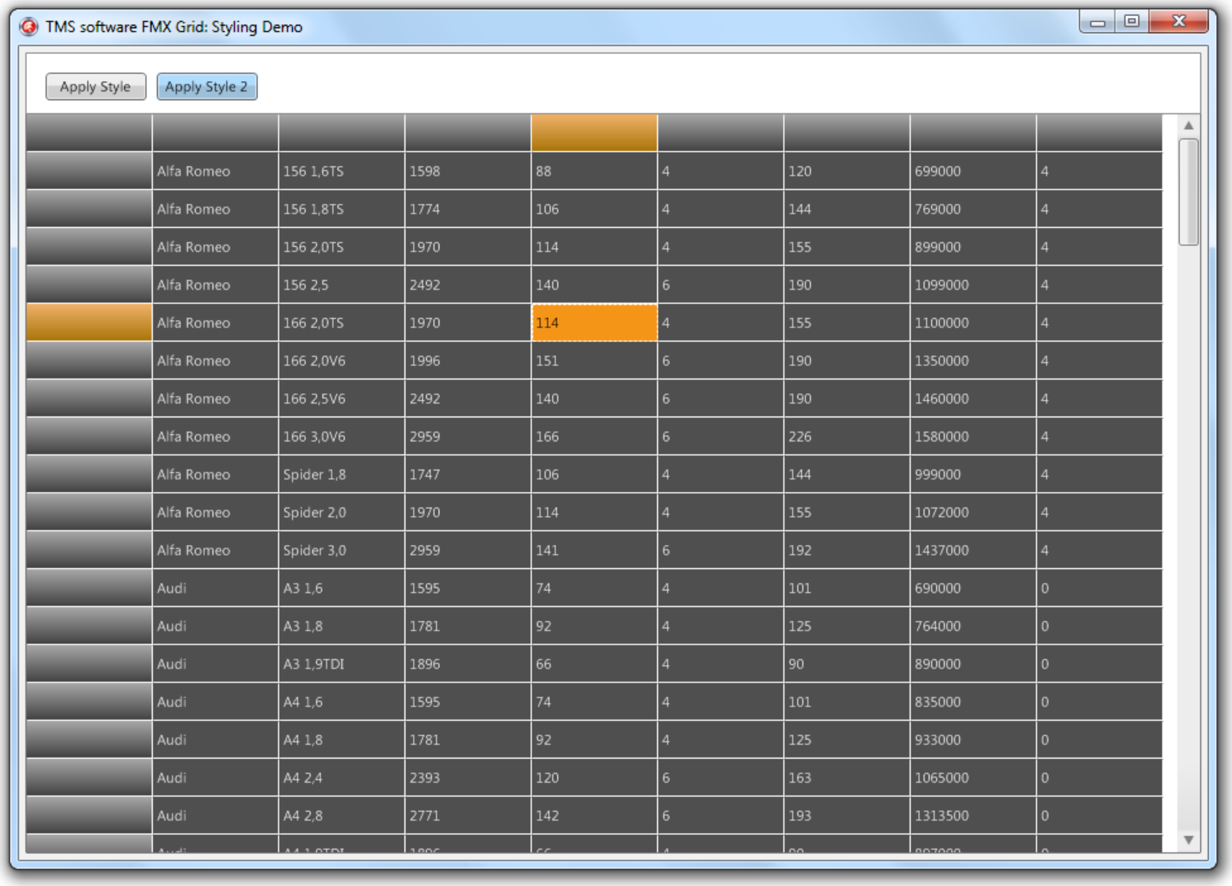
JAGUAR	XK8 cabrio	3996	209	8	280	2935000	1
	Diablo SV	5707	368	12	500	7410000	4
LAMBORGHINI	Diablo 2WD	5707	272	12	492	7720000	4
	Diablo VT	5707	272	12	492	8822599	4
LOTUS	Elise	1796	88	4	120	1175000	1
LOTUS	Esprit V8-GT	3506	260	8	354	2590000	1
	Ghibli Coupe 2,0	1996	225	6	306	2035825	4
		2790	209	6	284	2053975	4
MASERATI	3200 GT	3217	271	8	370	3126640	4
	Quattroporte 2,0	1996	211	6	287	2265725	4
	Quattroporte 2,8	2790	209	6	284	2459125	4
	Quattroporte 3,2	3217	247	8	335	3003825	4
MERCEDES	CLK 200	1998	100	4	136	1268000	0
MERCEDES	CLK 230	2295	142	4	193	1448000	0
MERCEDES	CLK 320	3199	160	6	218	1768000	0
MERCEDES	CLK 430	4266	205	8	279	2245000	0
MERCEDES	CLK cabrio 200	1998	100	4	136	1498000	0
MERCEDES	CLK cabrio 230	2295	142	4	193	1667000	0
MERCEDES	CLK cabrio 320	3199	160	6	218	1990000	0

Cells can be merged and splitted in the grid. Select a range of cells and press CTRL+M or CTRL+S to merge or split.

Also programmatically a selection of cells can be merged or splitted with:

```
TMSFMXGrid1.MergeSelection(TMSFMXGrid1.Selection);
TMSFMXGrid1.SplitCell(TMSFMXGrid1.FocusedCell.Col,TMSFMXGrid1.FocusedC
ell.Row);
```

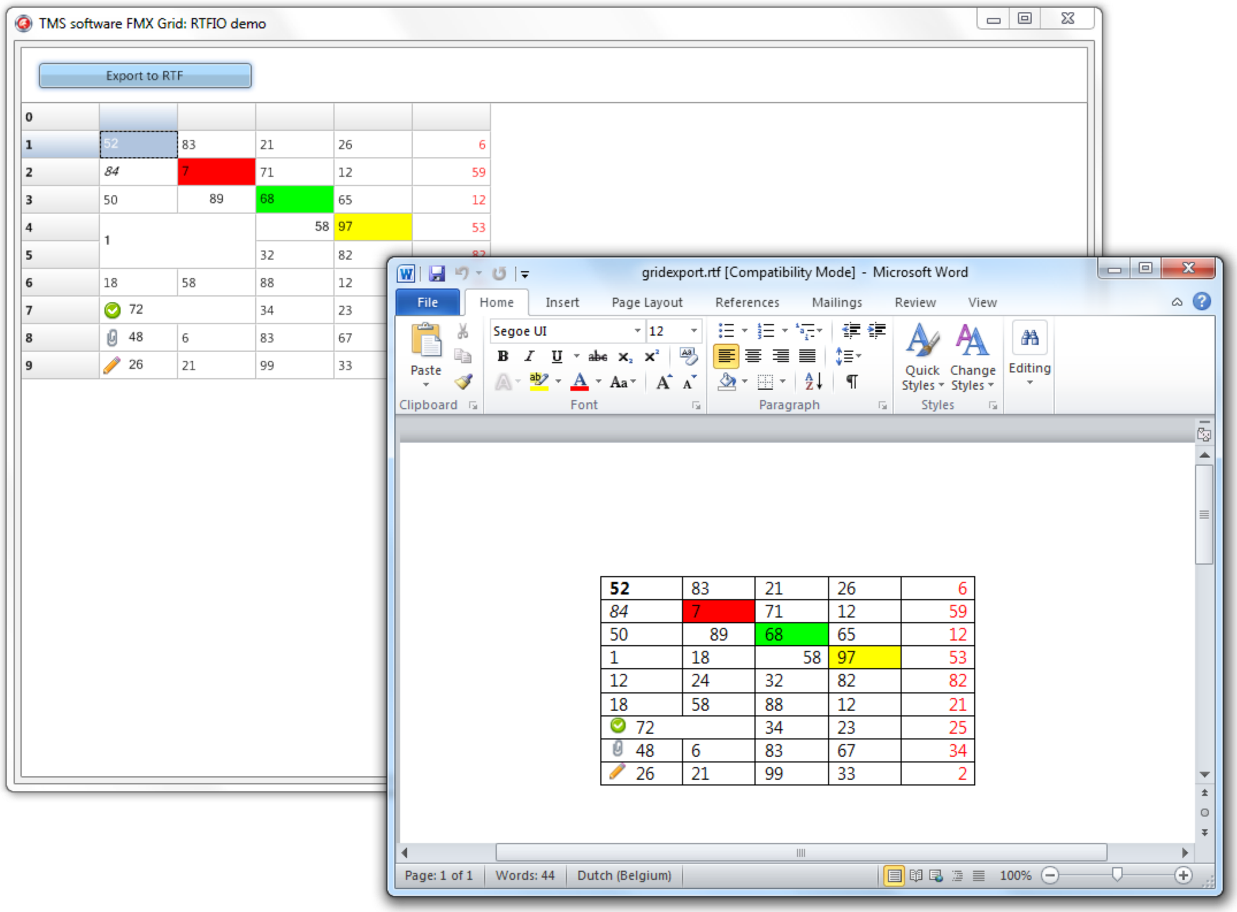
StylingDemo



	Alfa Romeo	156 1,6TS	1598	88	4	120	699000	4
	Alfa Romeo	156 1,8TS	1774	106	4	144	769000	4
	Alfa Romeo	156 2,0TS	1970	114	4	155	899000	4
	Alfa Romeo	156 2,5	2492	140	6	190	1099000	4
	Alfa Romeo	166 2,0TS	1970	114	4	155	1100000	4
	Alfa Romeo	166 2,0V6	1996	151	6	190	1350000	4
	Alfa Romeo	166 2,5V6	2492	140	6	190	1460000	4
	Alfa Romeo	166 3,0V6	2959	166	6	226	1580000	4
	Alfa Romeo	Spider 1,8	1747	106	4	144	999000	4
	Alfa Romeo	Spider 2,0	1970	114	4	155	1072000	4
	Alfa Romeo	Spider 3,0	2959	141	6	192	1437000	4
	Audi	A3 1,6	1595	74	4	101	690000	0
	Audi	A3 1,8	1781	92	4	125	764000	0
	Audi	A3 1,9TDI	1896	66	4	90	890000	0
	Audi	A4 1,6	1595	74	4	101	835000	0
	Audi	A4 1,8	1781	92	4	125	933000	0
	Audi	A4 2,4	2393	120	6	163	1065000	0
	Audi	A4 2,8	2771	142	6	193	1313500	0

The styling demo demonstrates changing different layouts. To see how this is achieved, open the Demo form in designtime and double-click on of the StyleBooks which contain the different layouts for the grid. There you will notice a style layout of the grid, with modified cell appearances.

RTFIODemo


TMS software FMX Grid: RTFIO demo
Export to RTF
gridexport.rtf [Compatibility Mode] - Microsoft Word
Page: 1 of 1
Words: 44
Dutch (Belgium)

52	83	21	26	6
84	7	71	12	59
50	89	68	65	12
1	18	58	97	53
12	24	32	82	82
18	58	88	12	21
✓ 72		34	23	25
48	6	83	67	34
26	21	99	33	2

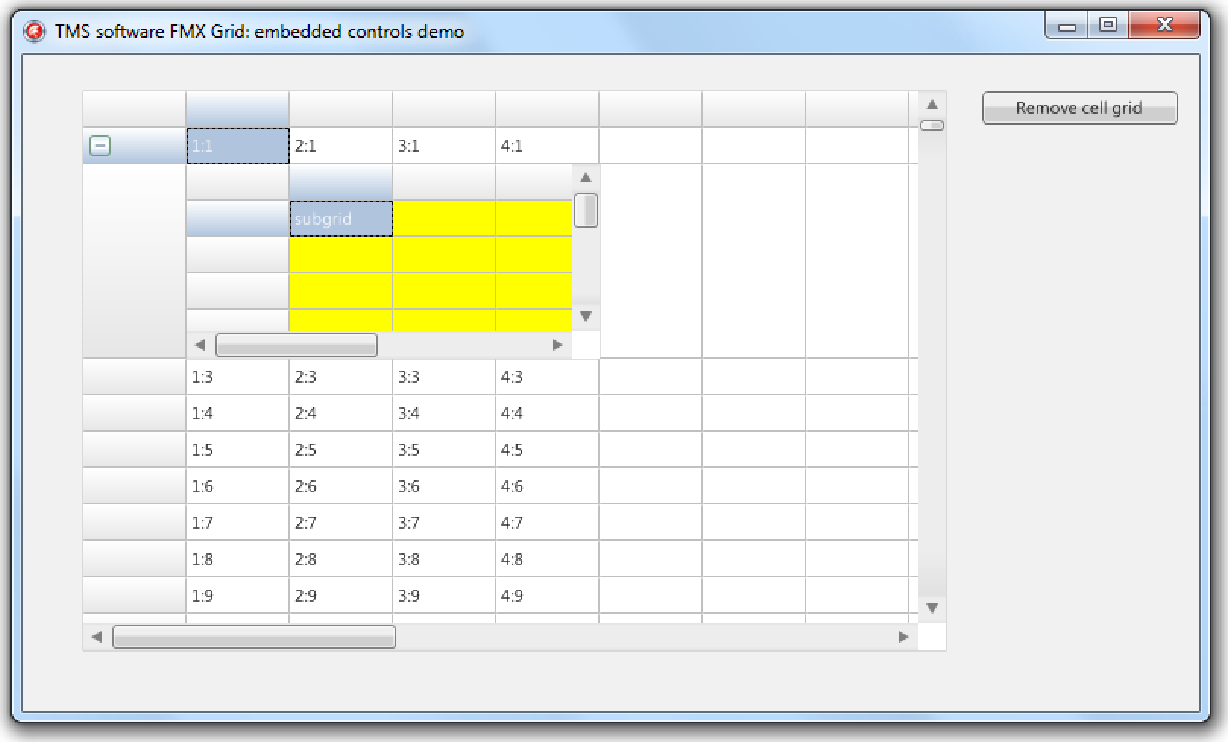
The RTF IO demo demonstrates exporting to and importing from an rich text compatible format. When clicking on the Export to RTF button, the grid is exported and the rtf file is automatically opened if a rich text editor is available.

Exporting the grid can be done with

```
TMSFMXGridRTFIO1.ExportRTF('..\..\gridexport.rtf');
XOpenFile('open', '..\..\gridexport.rtf', '', '');
```

Note that for opening the file, the XOpenFile procedure is used that is located in the unit FMX.TMSXUtil which is compatible with all supported platforms which can handle file manipulation.

EmbeddedControlsDemo



1:1	2:1	3:1	4:1			
1:3	2:3	3:3	4:3			
1:4	2:4	3:4	4:4			
1:5	2:5	3:5	4:5			
1:6	2:6	3:6	4:6			
1:7	2:7	3:7	4:7			
1:8	2:8	3:8	4:8			
1:9	2:9	3:9	4:9			

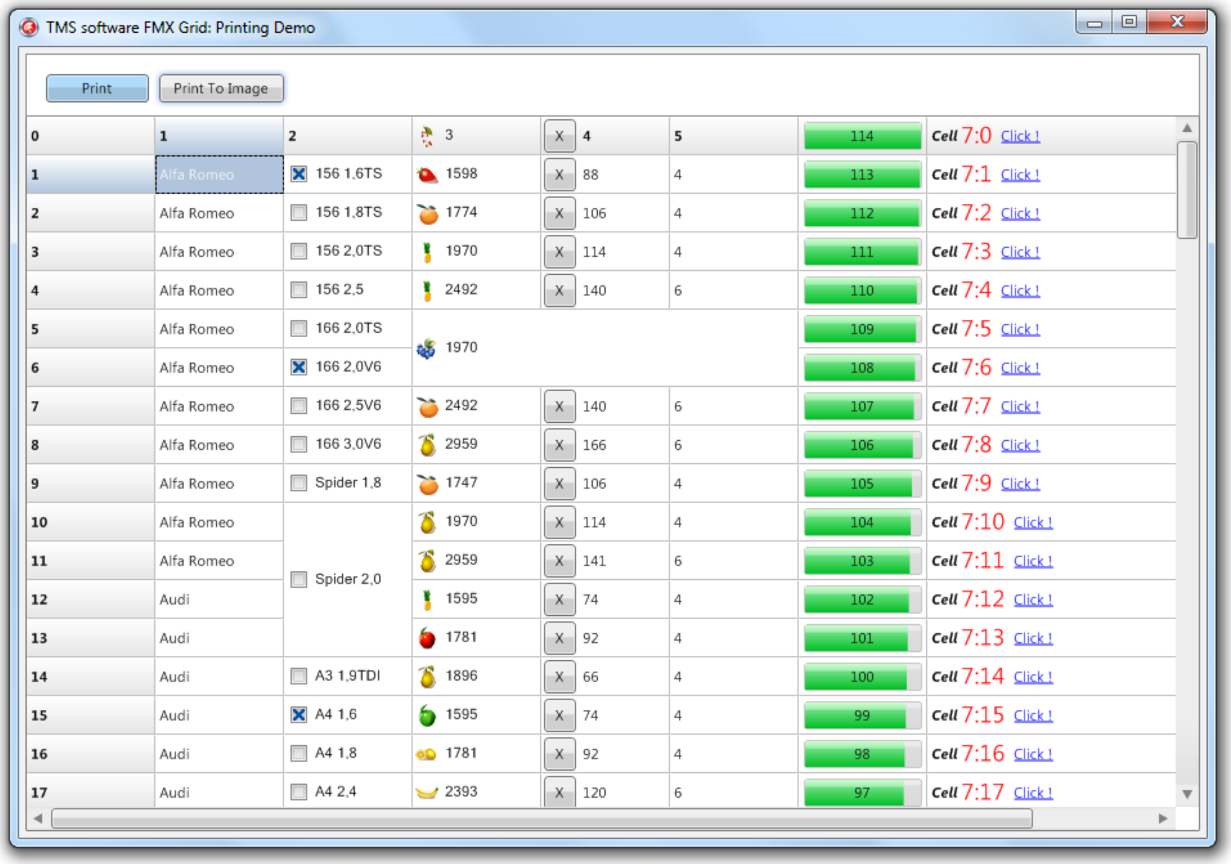
The Embedded Controls demo shows you how to add a control to a cell that is handled outside the grid.

With

```
TMSFMXGrid1.CellControls[ACol, ARow] := AControl;
```

You are able to add an additional control inside a gridcell.
The control is client-aligned, so with ColumnWidths and RowHeights, or cell merging the cell can be made larger so the control can be made visible and interactable.

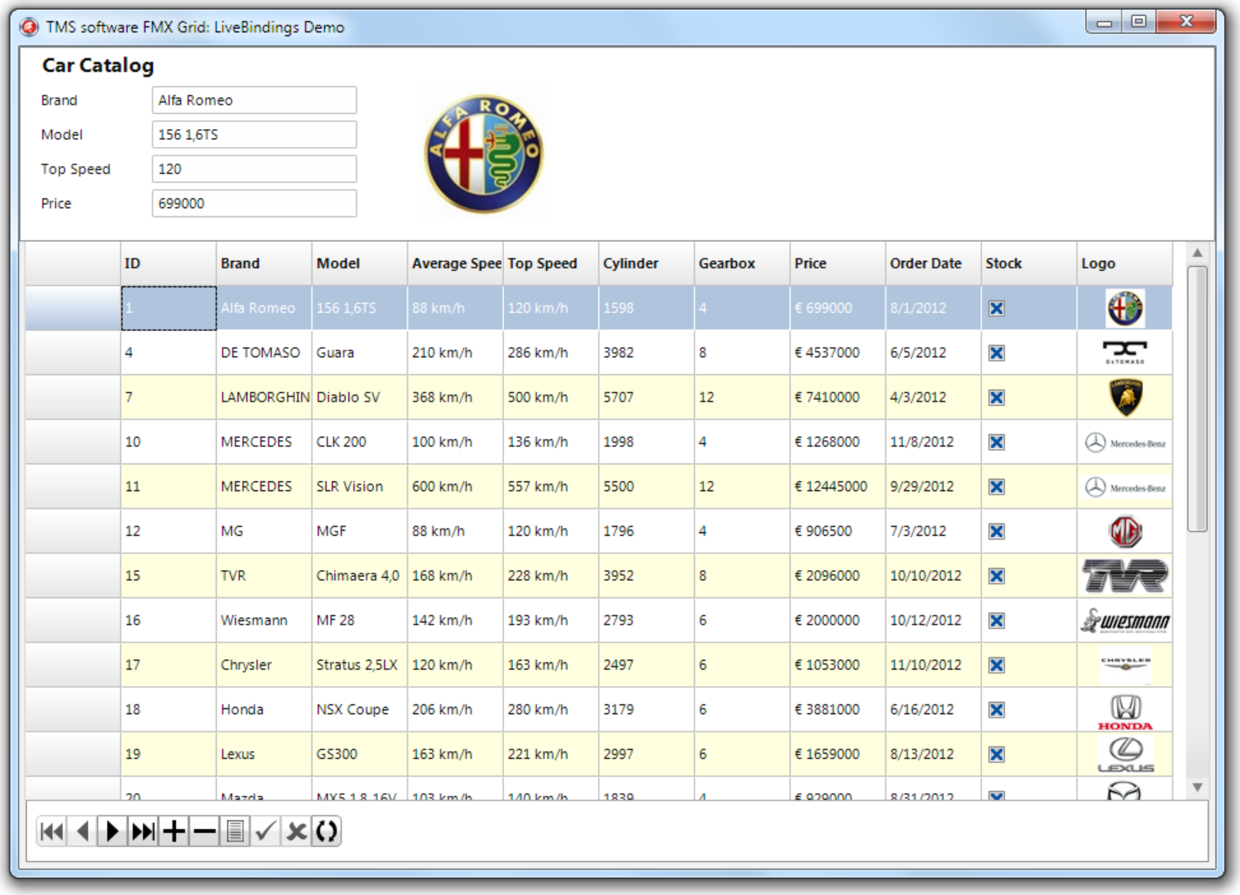
PrintingDemo



0	1	2	3	X 4	5	114	Cell 7:0 Click !
1	Alfa Romeo	☒ 156 1,6TS	1598	X 88	4	113	Cell 7:1 Click !
2	Alfa Romeo	☐ 156 1,8TS	1774	X 106	4	112	Cell 7:2 Click !
3	Alfa Romeo	☐ 156 2,0TS	1970	X 114	4	111	Cell 7:3 Click !
4	Alfa Romeo	☐ 156 2,5	2492	X 140	6	110	Cell 7:4 Click !
5	Alfa Romeo	☐ 166 2,0TS	1970			109	Cell 7:5 Click !
6	Alfa Romeo	☒ 166 2,0V6				108	Cell 7:6 Click !
7	Alfa Romeo	☐ 166 2,5V6	2492	X 140	6	107	Cell 7:7 Click !
8	Alfa Romeo	☐ 166 3,0V6	2959	X 166	6	106	Cell 7:8 Click !
9	Alfa Romeo	☐ Spider 1,8	1747	X 106	4	105	Cell 7:9 Click !
10	Alfa Romeo	☐ Spider 2,0	1970	X 114	4	104	Cell 7:10 Click !
11	Alfa Romeo		2959	X 141	6	103	Cell 7:11 Click !
12	Audi		1595	X 74	4	102	Cell 7:12 Click !
13	Audi		1781	X 92	4	101	Cell 7:13 Click !
14	Audi	☐ A3 1,9TDI	1896	X 66	4	100	Cell 7:14 Click !
15	Audi	☒ A4 1,6	1595	X 74	4	99	Cell 7:15 Click !
16	Audi	☐ A4 1,8	1781	X 92	4	98	Cell 7:16 Click !
17	Audi	☐ A4 2,4	2393	X 120	6	97	Cell 7:17 Click !

The printing demo demonstrates how to print to the printer and to an image. Printing supports several options and events and also supports printing to a separate canvas. When clicking on the print button, the printpreview dialog is shown that allows you to navigate through the pages and select the current page or a range of pages to print.

LiveBindings demo


TMS software FMX Grid: LiveBindings Demo

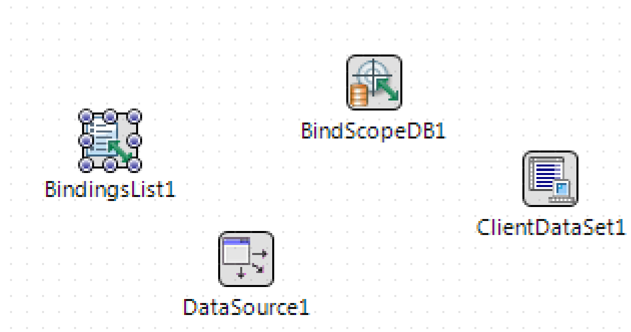
Car Catalog

Brand	Alfa Romeo
Model	156 1,6TS
Top Speed	120
Price	699000

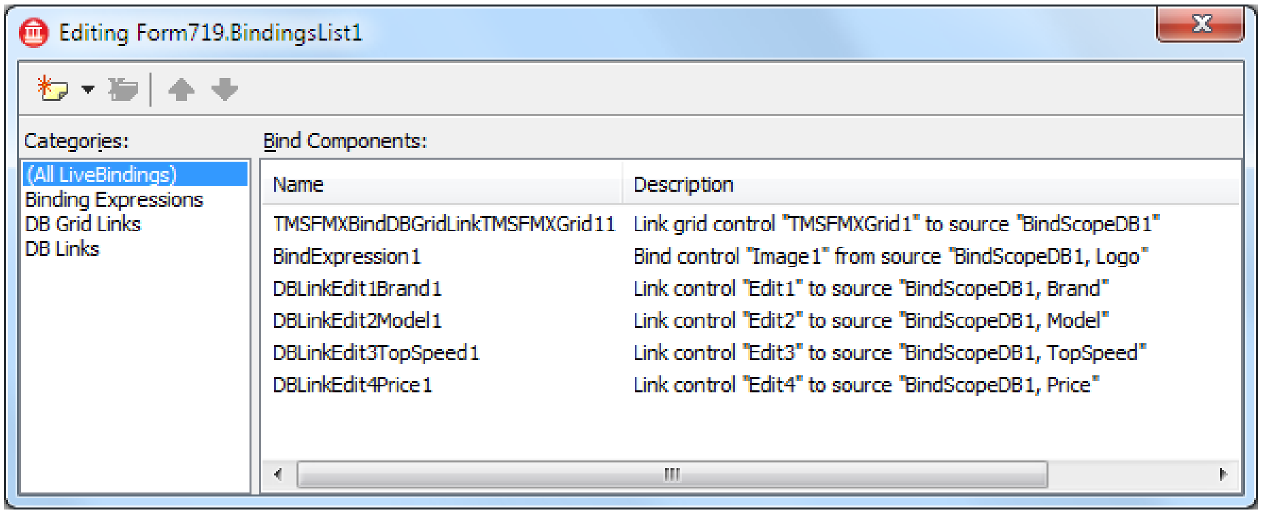
ID	Brand	Model	Average Spee	Top Speed	Cylinder	Gearbox	Price	Order Date	Stock	Logo
1	Alfa Romeo	156 1,6TS	88 km/h	120 km/h	1598	4	€ 699000	8/1/2012	☒	
4	DE TOMASO	Guara	210 km/h	286 km/h	3982	8	€ 4537000	6/5/2012	☒	
7	LAMBORGHIN	Diablo SV	368 km/h	500 km/h	5707	12	€ 7410000	4/3/2012	☒	
10	MERCEDES	CLK 200	100 km/h	136 km/h	1998	4	€ 1268000	11/8/2012	☒	
11	MERCEDES	SLR Vision	600 km/h	557 km/h	5500	12	€ 12445000	9/29/2012	☒	
12	MG	MGF	88 km/h	120 km/h	1796	4	€ 906500	7/3/2012	☒	
15	TVR	Chimaera 4,0	168 km/h	228 km/h	3952	8	€ 2096000	10/10/2012	☒	
16	Wiesmann	MF 28	142 km/h	193 km/h	2793	6	€ 2000000	10/12/2012	☒	
17	Chrysler	Stratus 2,5LX	120 km/h	163 km/h	2497	6	€ 1053000	11/10/2012	☒	
18	Honda	NSX Coupe	206 km/h	280 km/h	3179	6	€ 3881000	6/16/2012	☒	
19	Lexus	GS300	163 km/h	221 km/h	2997	6	€ 1659000	8/13/2012	☒	
20	Mazda	MX5 1.8 16V	103 km/h	140 km/h	1839	4	€ 929000	8/31/2012	☒	

This demo is a LiveBindings sample, specifically designed in combination with the TMSFMXGrid. The demo loads a ClientDataSet with a cars.xml sample data file. The ClientDataSet is then connected, in combination with a DataSource, to a BindScope, which is needed to bind data to the ListBindings component. You can choose to manually bind the data to the grid, or automatically. Both are explained here below.

<u>Manually</u>



The demo makes use of a TTMSFMXBindDBGridLink that is created and registered specifically for the Grid component. When opening the editor of the BindingsList you will see various bindings to different elements that are placed on the form. All these elements have a binding to a specific field and will update when navigating through the list or with the BindNavigator component.


Editing Form719.BindingsList1

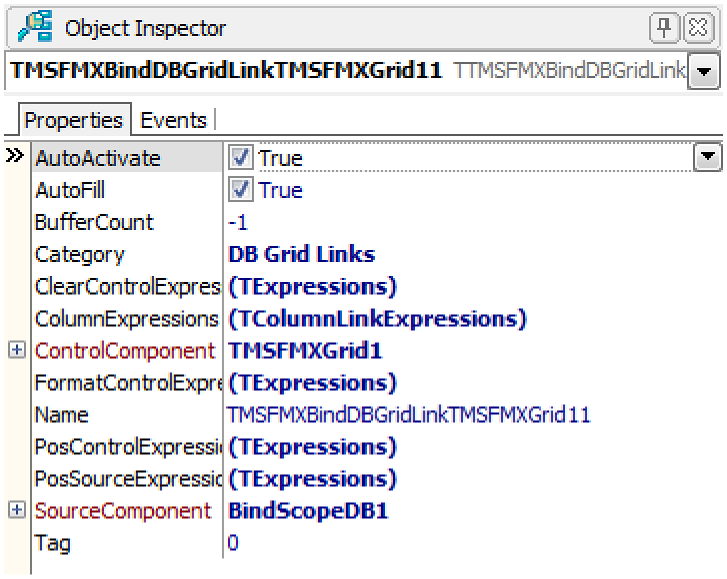
Categories:

- (All LiveBindings)
- Binding Expressions
- DB Grid Links
- DB Links

Bind Components:

Name	Description
TMSFMXBindDBGridLinkTMSFMXGrid11	Link grid control "TMSFMXGrid1" to source "BindScopeDB1"
BindExpression1	Bind control "Image1" from source "BindScopeDB1, Logo"
DBLinkEdit1Brand1	Link control "Edit1" to source "BindScopeDB1, Brand"
DBLinkEdit2Model1	Link control "Edit2" to source "BindScopeDB1, Model"
DBLinkEdit3TopSpeed1	Link control "Edit3" to source "BindScopeDB1, TopSpeed"
DBLinkEdit4Price1	Link control "Edit4" to source "BindScopeDB1, Price"

Selecting the TMSFMXBindDBGridLinkTMSFMXGrid11 component will display its properties in the object inspector.



Object Inspector

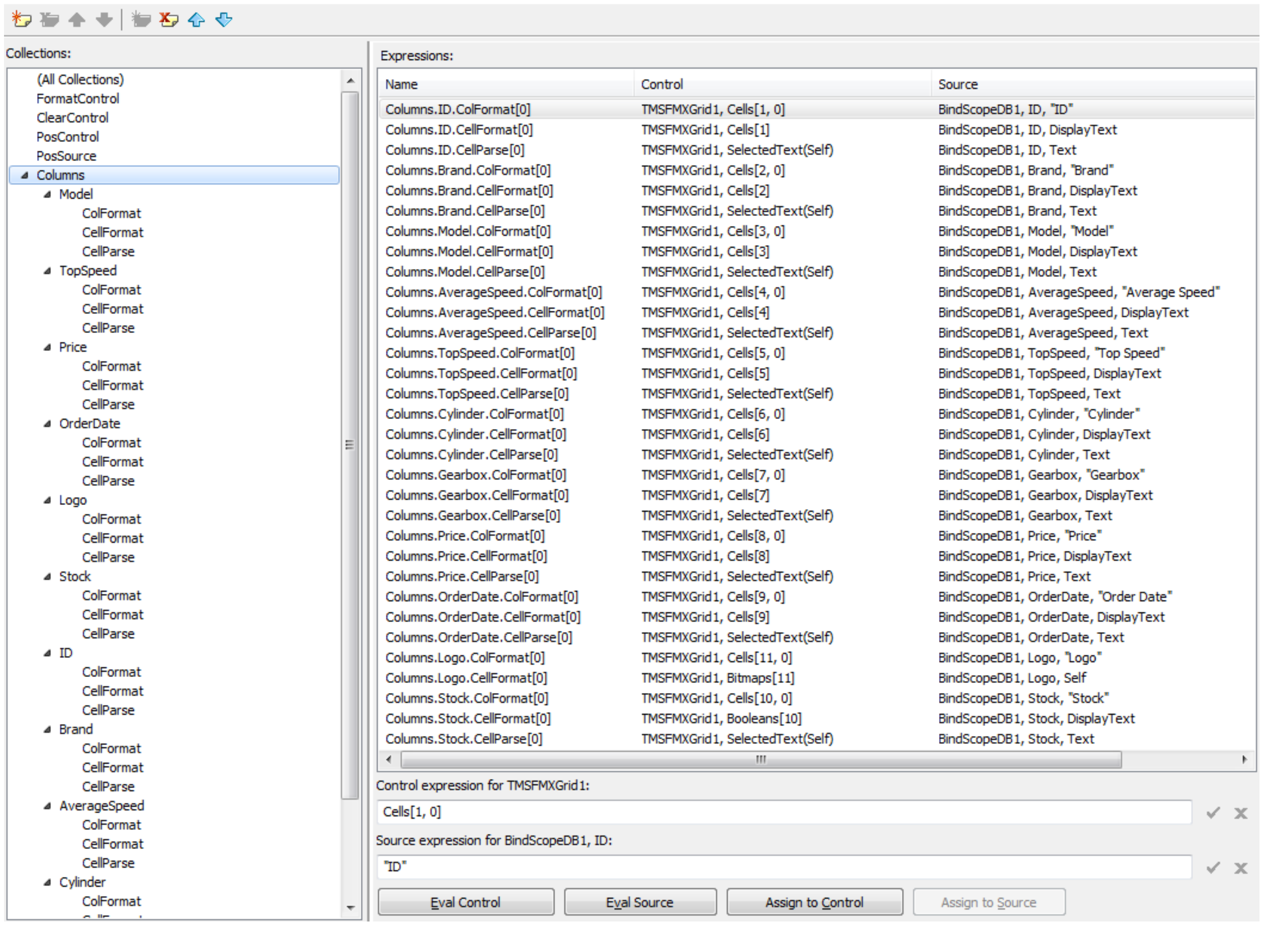
TMSFMXBindDBGridLinkTMSFMXGrid11 TTMSFMXBindDBGridLink

Properties | Events

AutoActivate	✓ True
AutoFill	✓ True
BufferCount	-1
Category	DB Grid Links
ClearControlExpres	(TExpressions)
ColumnExpressions	(TColumnLinkExpressions)
ControlComponent	TMSFMXGrid1
FormatControlExpre	(TExpressions)
Name	TMSFMXBindDBGridLinkTMSFMXGrid11
PosControlExpressi	(TExpressions)
PosSourceExpressic	(TExpressions)
SourceComponent	BindScopeDB1
Tag	0

As with the TrackBar sample in the LiveBindings chapter this link has a SourceComponent and a ControlComponent. The difference between this link and the link in the TrackBar sample is, that the TMSFMXBindDBGridLink component is able to bind multiple fields to multiple columns in the grid.

In the sample you will notice that there is binding to the Fields that are loaded from the clientdataset located in the cars.xml file. To bind data to these elements, an expression must be added per field to the ColumnExpressions collection. Double-clicking on the ColumnExpressions property opens the expressions editor.



- (All Collections)
- FormatControl
- ClearControl
- PosControl
- PosSource
- Columns
  - Model
    - ColFormat
    - CellFormat
    - CellParse
  - TopSpeed
    - ColFormat
    - CellFormat
    - CellParse
  - Price
    - ColFormat
    - CellFormat
    - CellParse
  - OrderDate
    - ColFormat
    - CellFormat
    - CellParse
  - Logo
    - ColFormat
    - CellFormat
    - CellParse
  - Stock
    - ColFormat
    - CellFormat
    - CellParse
  - ID
    - ColFormat
    - CellFormat
    - CellParse
  - Brand
    - ColFormat
    - CellFormat
    - CellParse
  - AverageSpeed
    - ColFormat
    - CellFormat
    - CellParse
  - Cylinder
    - ColFormat

Name	Control	Source
Columns.ID.ColFormat[0]	TMSFMXGrid1, Cells[1, 0]	BindScopeDB1, ID, "ID"
Columns.ID.CellFormat[0]	TMSFMXGrid1, Cells[1]	BindScopeDB1, ID, DisplayText
Columns.ID.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindScopeDB1, ID, Text
Columns.Brand.ColFormat[0]	TMSFMXGrid1, Cells[2, 0]	BindScopeDB1, Brand, "Brand"
Columns.Brand.CellFormat[0]	TMSFMXGrid1, Cells[2]	BindScopeDB1, Brand, DisplayText
Columns.Brand.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindScopeDB1, Brand, Text
Columns.Model.ColFormat[0]	TMSFMXGrid1, Cells[3, 0]	BindScopeDB1, Model, "Model"
Columns.Model.CellFormat[0]	TMSFMXGrid1, Cells[3]	BindScopeDB1, Model, DisplayText
Columns.Model.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindScopeDB1, Model, Text
Columns.AverageSpeed.ColFormat[0]	TMSFMXGrid1, Cells[4, 0]	BindScopeDB1, AverageSpeed, "Average Speed"
Columns.AverageSpeed.CellFormat[0]	TMSFMXGrid1, Cells[4]	BindScopeDB1, AverageSpeed, DisplayText
Columns.AverageSpeed.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindScopeDB1, AverageSpeed, Text
Columns.TopSpeed.ColFormat[0]	TMSFMXGrid1, Cells[5, 0]	BindScopeDB1, TopSpeed, "Top Speed"
Columns.TopSpeed.CellFormat[0]	TMSFMXGrid1, Cells[5]	BindScopeDB1, TopSpeed, DisplayText
Columns.TopSpeed.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindScopeDB1, TopSpeed, Text
Columns.Cylinder.ColFormat[0]	TMSFMXGrid1, Cells[6, 0]	BindScopeDB1, Cylinder, "Cylinder"
Columns.Cylinder.CellFormat[0]	TMSFMXGrid1, Cells[6]	BindScopeDB1, Cylinder, DisplayText
Columns.Cylinder.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindScopeDB1, Cylinder, Text
Columns.Gearbox.ColFormat[0]	TMSFMXGrid1, Cells[7, 0]	BindScopeDB1, Gearbox, "Gearbox"
Columns.Gearbox.CellFormat[0]	TMSFMXGrid1, Cells[7]	BindScopeDB1, Gearbox, DisplayText
Columns.Gearbox.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindScopeDB1, Gearbox, Text
Columns.Price.ColFormat[0]	TMSFMXGrid1, Cells[8, 0]	BindScopeDB1, Price, "Price"
Columns.Price.CellFormat[0]	TMSFMXGrid1, Cells[8]	BindScopeDB1, Price, DisplayText
Columns.Price.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindScopeDB1, Price, Text
Columns.OrderDate.ColFormat[0]	TMSFMXGrid1, Cells[9, 0]	BindScopeDB1, OrderDate, "Order Date"
Columns.OrderDate.CellFormat[0]	TMSFMXGrid1, Cells[9]	BindScopeDB1, OrderDate, DisplayText
Columns.OrderDate.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindScopeDB1, OrderDate, Text
Columns.Logo.ColFormat[0]	TMSFMXGrid1, Cells[11, 0]	BindScopeDB1, Logo, "Logo"
Columns.Logo.CellFormat[0]	TMSFMXGrid1, Bitmaps[11]	BindScopeDB1, Logo, Self
Columns.Stock.ColFormat[0]	TMSFMXGrid1, Cells[10, 0]	BindScopeDB1, Stock, "Stock"
Columns.Stock.CellFormat[0]	TMSFMXGrid1, Booleans[10]	BindScopeDB1, Stock, DisplayText
Columns.Stock.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindScopeDB1, Stock, Text

Control expression for TMSFMXGrid1:
Cells[1, 0]

Source expression for BindScopeDB1, ID:
"ID"

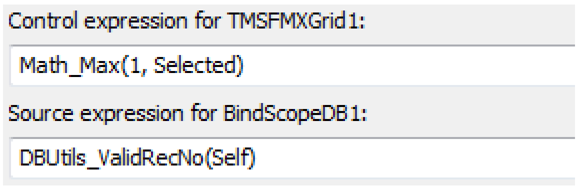
For the ID, the binding has been added to an integer field in the DataBase. To return the value for the current record, the AsInteger function must be used in the Source expression. The ColFormat is used to display the header in the fixed row of the grid. The CellFormat is used to display the cell data depending on the active row. The CellParse is used to detect the selected text when editing is done. The CellParse expression looks identically for each column. The CellFormat is the most important expression that is used to insert the data in the grid.

As you will notice, the most used control expression in either ColFormat / CellFormat or CellParse is the Cells[] property. This is also available in a standard grid. The Cells property with one value passed through is used internally when the record pointer changes to load the data in the grid. In the ColFormat the Cells property is used with 2 parameters and will remain fixed during the runtime of the application.

To bind the Logo, the source expression is Self and the control expression is Bitmap[]. To know exactly which source expression you must use, you can click on Eval Source which will tell you the type of the data. To use the Eval Source function the DataSet must be active. Currently the grid supports 4 kinds of cell binding:

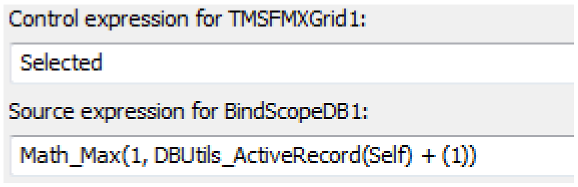
Cells[] : All text based data
Bitmaps[] : All graphical / blobfield type data
Booleans[] : All Boolean data.
Colors[]: All Color data (integer).
When starting the application, the list will load the data and display the different columns in the grid. When selecting a different row, the navigation in the dataset is automatically handled. This is due to the initialization of the TTMSFMXBindDBGridLink that has already taken care of this functionality and has stored the correct values in the PosSource and PosControl expressions.

PosSource:



Control expression for TMSFMXGrid1:
Math_Max(1, Selected)
Source expression for BindScopeDB1:
DBUtils_ValidRecNo(Self)

PosControl:



Control expression for TMSFMXGrid1:
Selected
Source expression for BindScopeDB1:
Math_Max(1, DBUtils_ActiveRecord(Self) + (1))

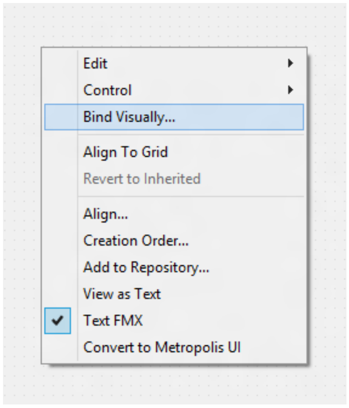
When double-clicking on a cell (based on the default settings in grid.options) the editing starts. When typing in the edit box, the dataset is automatically set in edit mode. When pressing enter the edit is stopped and the value is inserted in the cell. The value is not automatically added in the database, therefore a post must be done through the BindNavigator component. The value is also automatically posted when editing is still active and the post button has been pressed in on BindNavigator component.

The editor must be specified manually, there is no support to detect which fieldtype that has been added so the default editor is used on all field (etEdit). To specify a different editor for specific field you need to override the OnGetCellEditorType:

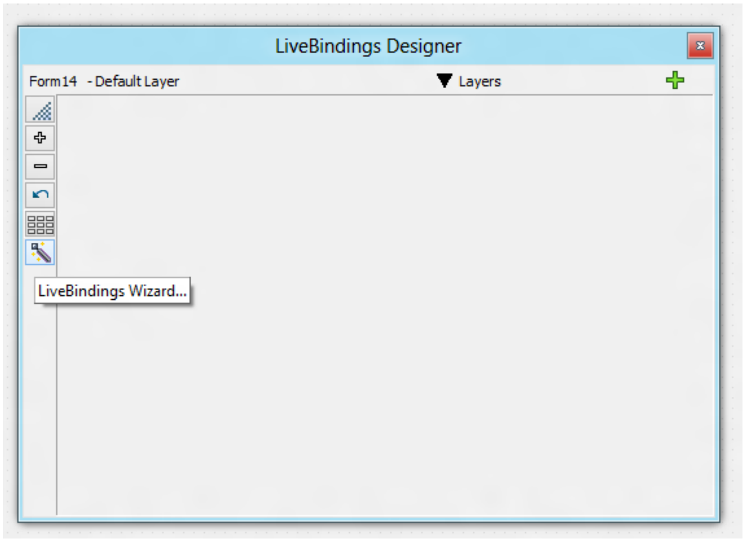
```
procedure TForm1.TMSFMXGrid1GetCellEditorType(Sender: TObject; ACol,
  ARow: Integer; var CellEditorType: TTMSFMXGridEditorType);
begin
  case ACol of
    7: CellEditorType := etNumericEdit;
    9: CellEditorType := etDateEdit;
  end;
end;
```

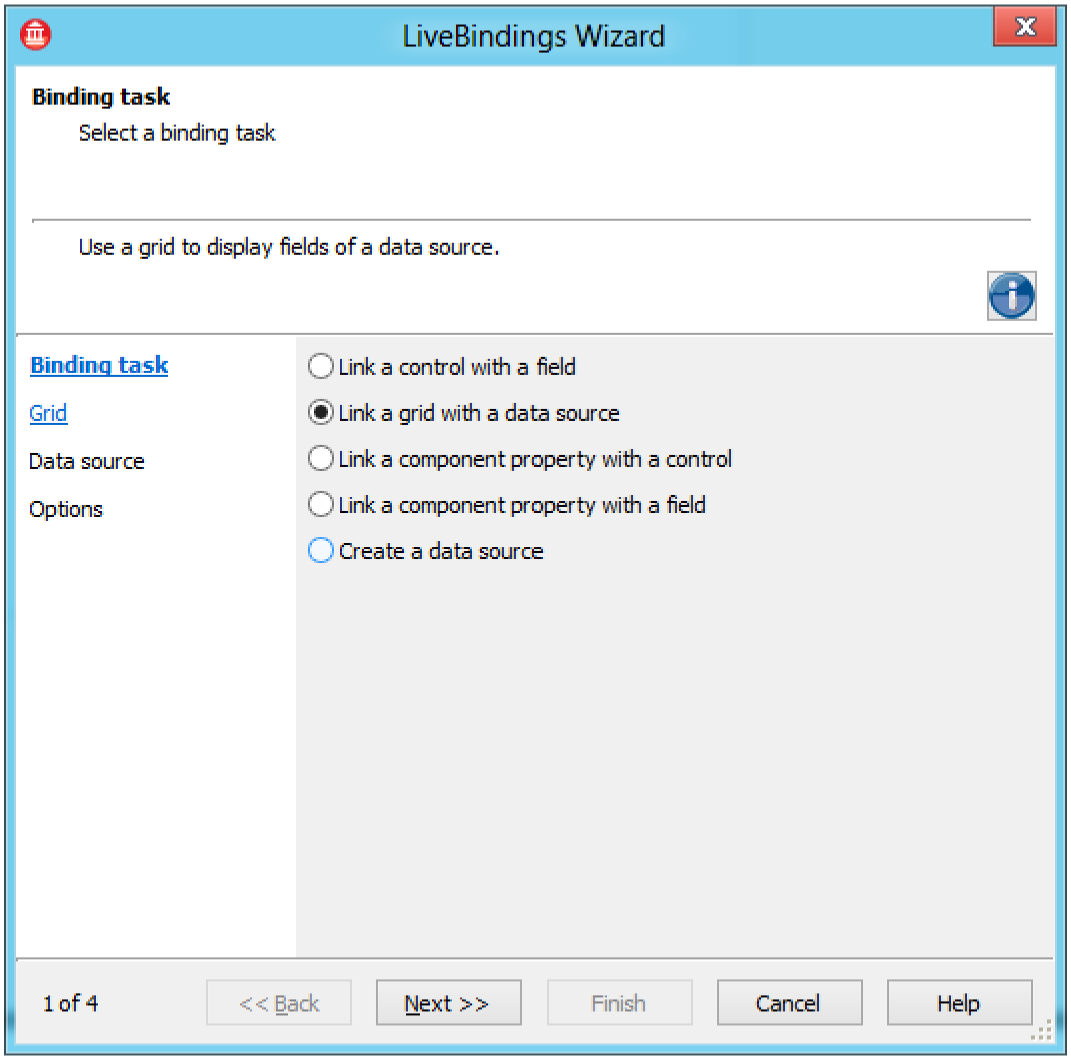
<u>Automatically</u>

Start a new application and right-click on the form. Select Bind Visually… to start the visual designer to create a livebinding application.



The designer shows a visual overview of the links between the components and datasources / other components. Click on the livebindings wizard icon to start the wizard.




LiveBindings Wizard

Binding task

Select a binding task

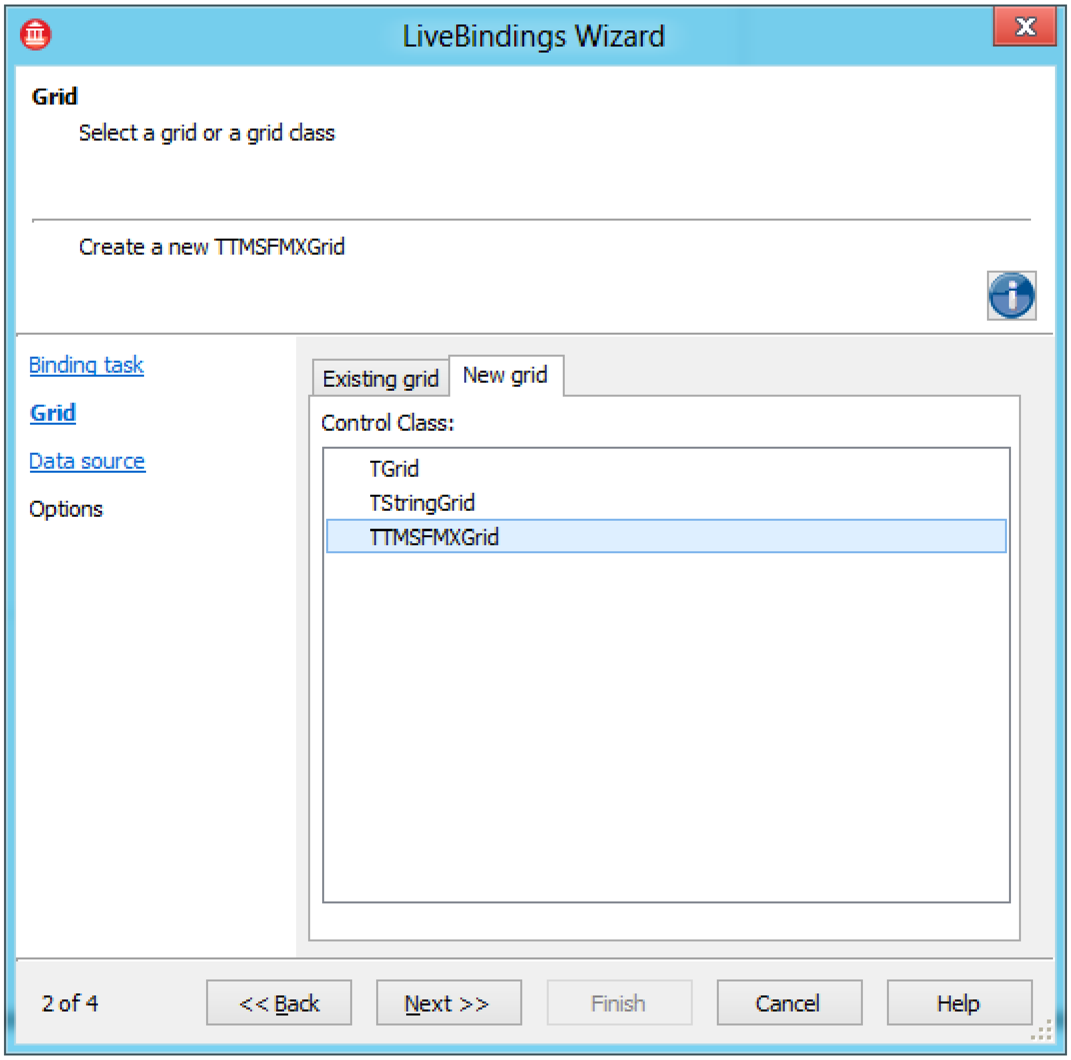
Use a grid to display fields of a data source.

- **Binding task**
- Grid
- Data source
- Options

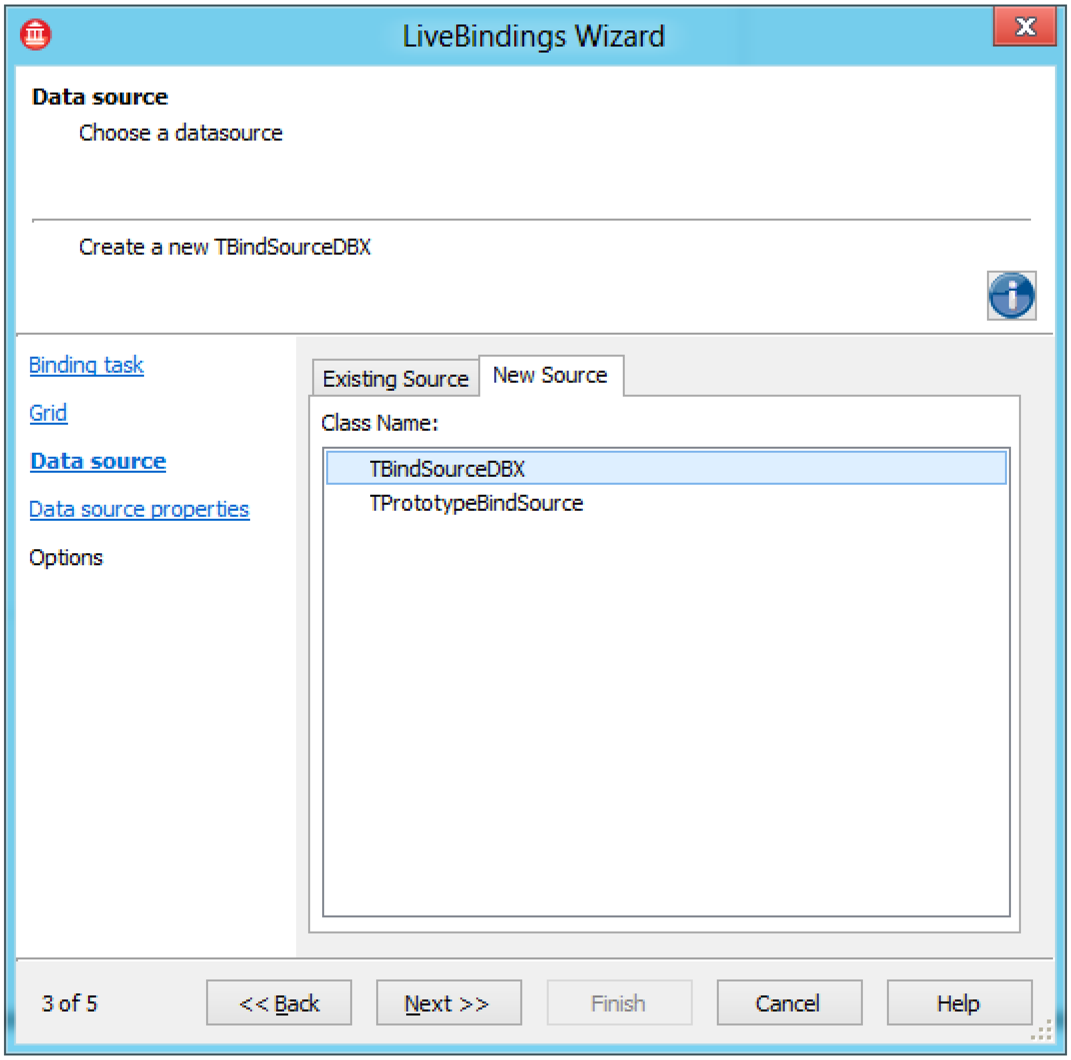
- ○ Link a control with a field
- ● Link a grid with a data source
- ○ Link a component property with a control
- ○ Link a component property with a field
- ○ Create a data source

1 of 4

You have multple options to choose from, either linking a component to a data source or linking a component to a control (see trackbar sample). For our grid, the Link a grid with a data source is marked. Click on the next button to go to the next page.



On this page the TTMSFMXGrid is available to be used as a new grid. On the tab Existing Grid you will also be able to choose an existing TTMSFMXGrid instance if you already have added a grid.



Data source

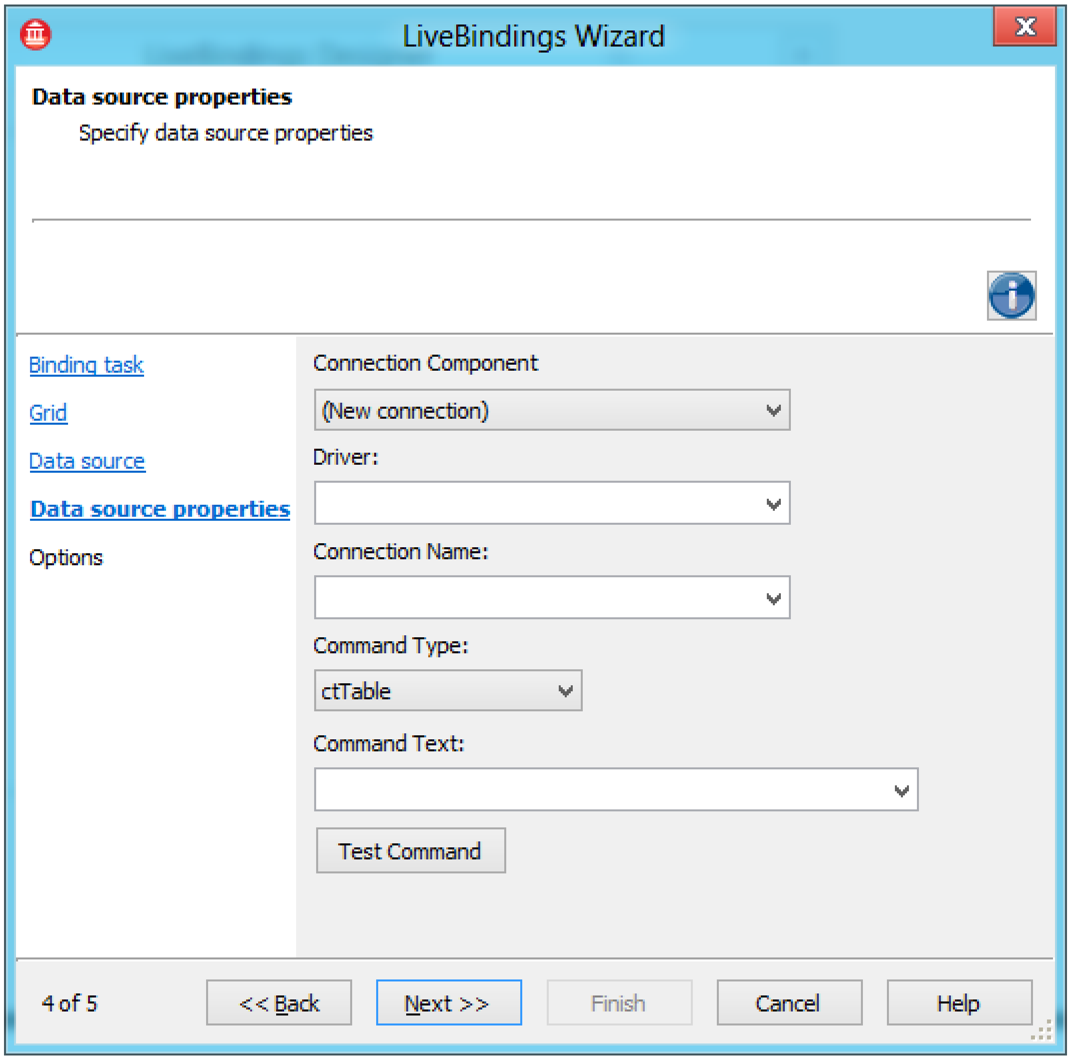
Choose a datasource

Create a new TBindSourceDBX

Class Name:

TBindSourceDBX
TPrototypeBindSource

The datasource page has 2 options, to bind the grid to a TBindSourceDBX, or to a TPrototypeBindSource. The TProtoTypeBindSource is a datasource that allows adding fields with dummy data to quickly create a test sample for your application. The bindsource can be easily replaced if real data is used. The real datasource is created with TBindSourceDBX. This has an extra page available that shows the datasource connection properties.



Data source properties

Specify data source properties

Connection Component: (New connection)

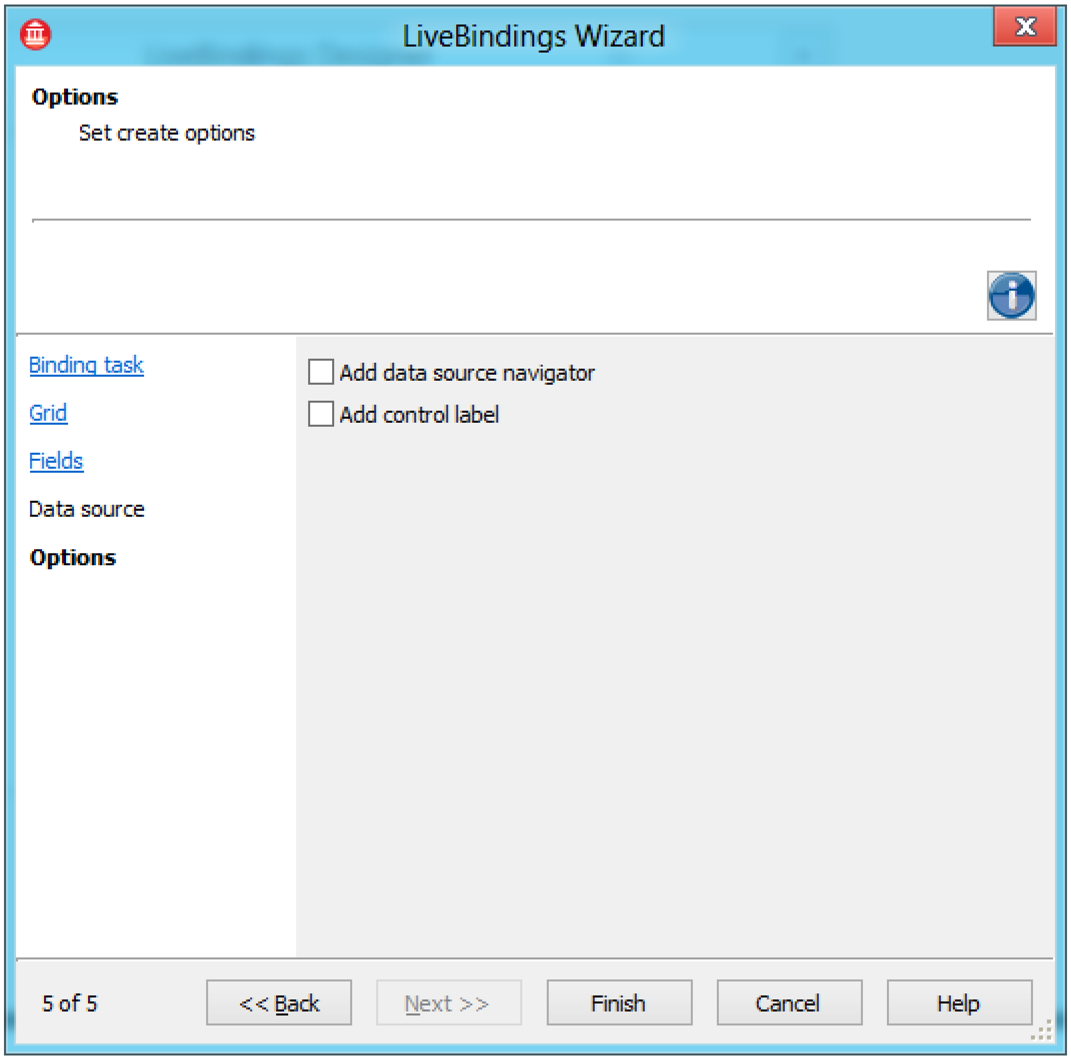
Driver:

Connection Name:

Command Type: ctTable

Command Text:

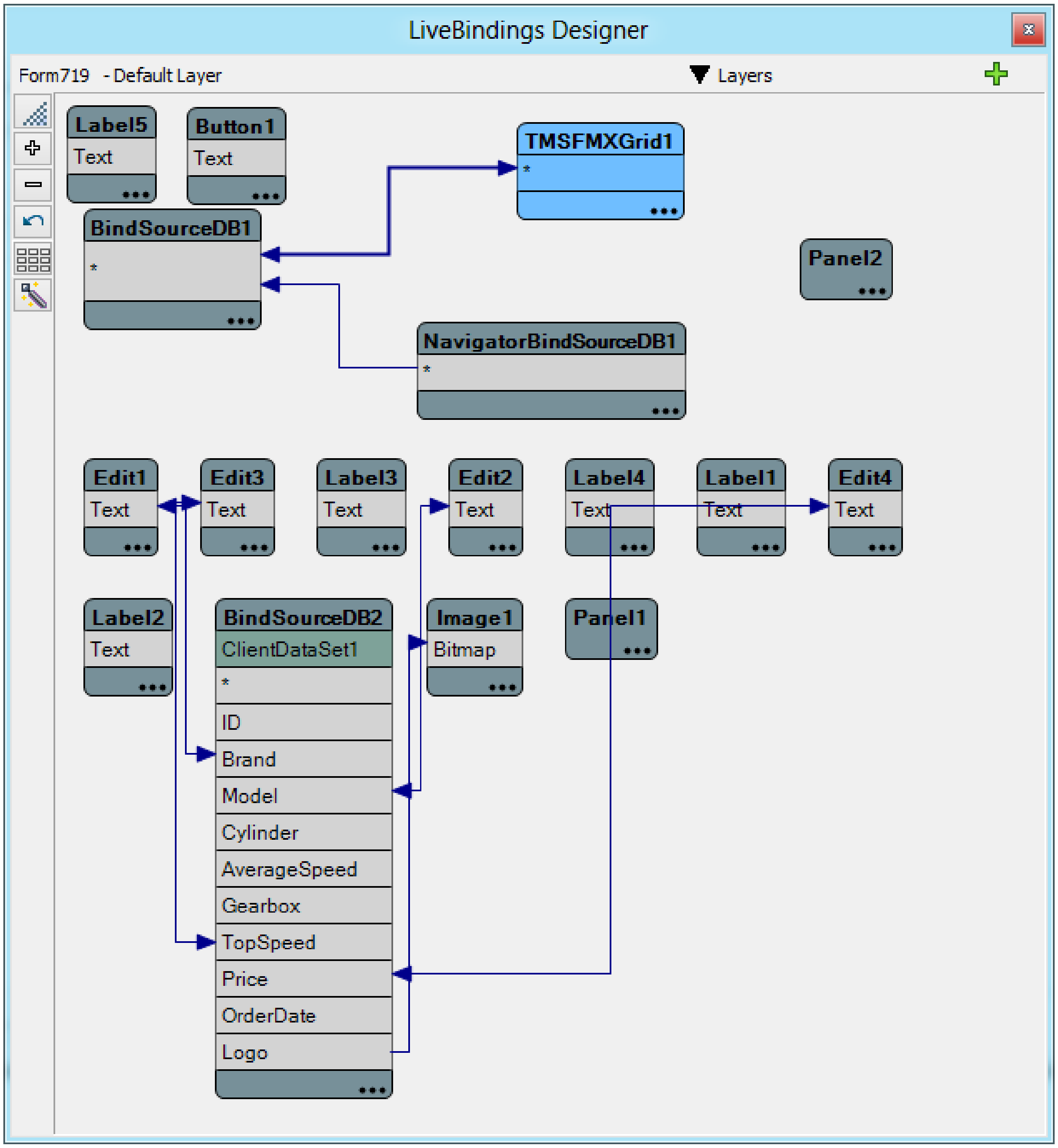
On the Data Source connection properties page you can specify which driver you wish to use among some other options such as a command type and a command text.



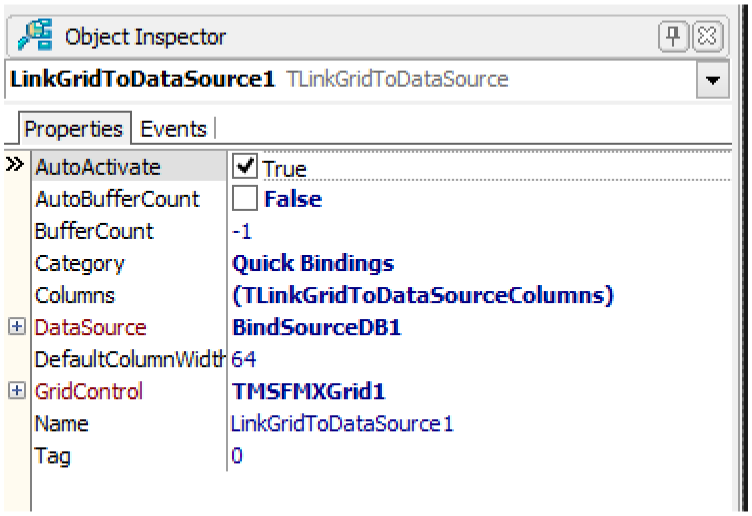
☐ Add data source navigator
☐ Add control label

The last page allows you to add a Data source navigator to navigate through the records.

When clicking on the finish button, the visual designer and components are updated showing your links you have created through the livebindings wizard. For this sample we show the visual designer that is linked to a ClientDataSet:



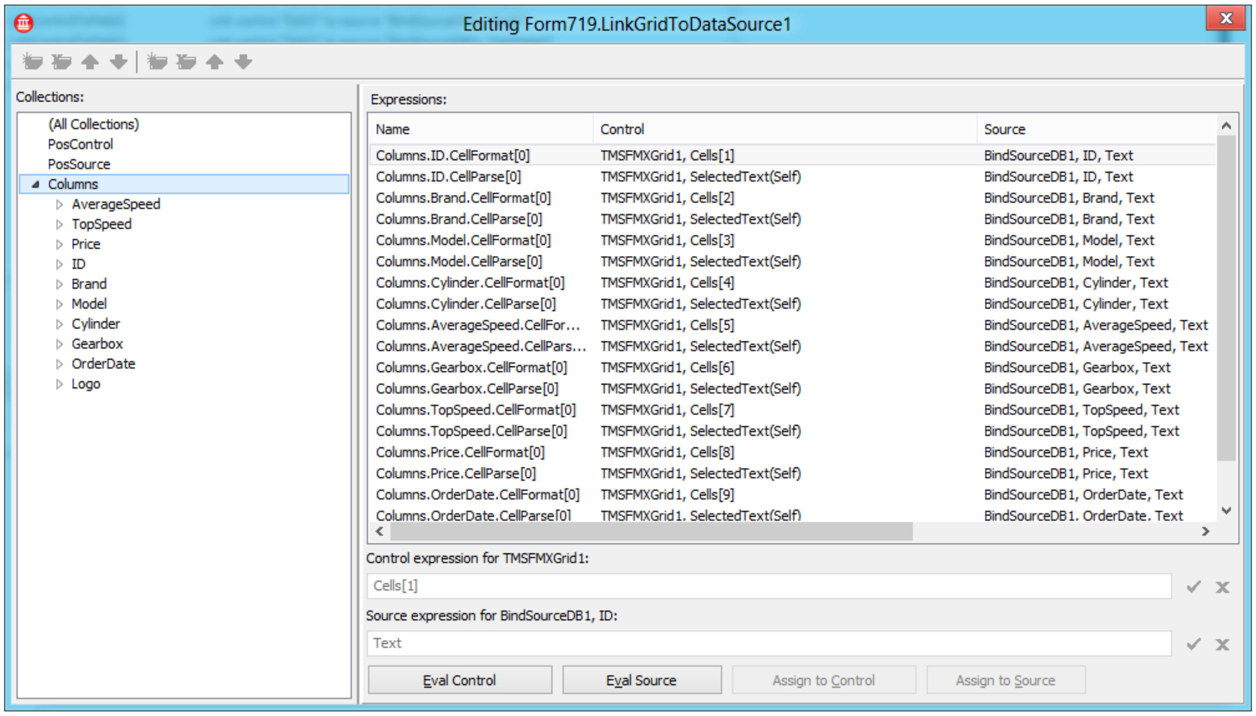
When selecting the BindingsList component and showing the list of Bindings, you will notice a LinkGridToDataSource link available that is created after finishing the LiveBindings wizard.



LinkGridToDataSource1 TLinkGridToDataSource

AutoActivate	✓ True
AutoBufferCount	False
BufferCount	-1
Category	Quick Bindings
Columns	(TLinkGridToDataSourceColumns)
DataSource	BindSourceDB1
DefaultColumnWidth	64
GridControl	TMSFMXGrid1
Name	LinkGridToDataSource1
Tag	0

This LinkGridToDataSource automatically loads the Fields and applies the correct Source and Control expression for the columns. When double-clicking on the LinkGridToDataSource in the BindingsList window, you will see that all expressions are added and cannot be modified. If you wish to modify an expression, such as a boolean field that needs to be presented as a text field, you need to work with the TTMSFMXBindDBGridLink that allows columns and expressions to be modified.



Name	Control	Source
Columns.ID.CellFormat[0]	TMSFMXGrid1, Cells[1]	BindSourceDB1, ID, Text
Columns.ID.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindSourceDB1, ID, Text
Columns.Brand.CellFormat[0]	TMSFMXGrid1, Cells[2]	BindSourceDB1, Brand, Text
Columns.Brand.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindSourceDB1, Brand, Text
Columns.Model.CellFormat[0]	TMSFMXGrid1, Cells[3]	BindSourceDB1, Model, Text
Columns.Model.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindSourceDB1, Model, Text
Columns.Cylinder.CellFormat[0]	TMSFMXGrid1, Cells[4]	BindSourceDB1, Cylinder, Text
Columns.Cylinder.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindSourceDB1, Cylinder, Text
Columns.AverageSpeed.CellFor...	TMSFMXGrid1, Cells[5]	BindSourceDB1, AverageSpeed, Text
Columns.AverageSpeed.CellPars...	TMSFMXGrid1, SelectedText(Self)	BindSourceDB1, AverageSpeed, Text
Columns.Gearbox.CellFormat[0]	TMSFMXGrid1, Cells[6]	BindSourceDB1, Gearbox, Text
Columns.Gearbox.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindSourceDB1, Gearbox, Text
Columns.TopSpeed.CellFormat[0]	TMSFMXGrid1, Cells[7]	BindSourceDB1, TopSpeed, Text
Columns.TopSpeed.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindSourceDB1, TopSpeed, Text
Columns.Price.CellFormat[0]	TMSFMXGrid1, Cells[8]	BindSourceDB1, Price, Text
Columns.Price.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindSourceDB1, Price, Text
Columns.OrderDate.CellFormat[0]	TMSFMXGrid1, Cells[9]	BindSourceDB1, OrderDate, Text
Columns.OrderDate.CellParse[0]	TMSFMXGrid1, SelectedText(Self)	BindSourceDB1, OrderDate, Text

Control expression for TMSFMXGrid1:
Cells[1]
Source expression for BindSourceDB1, ID:
Text

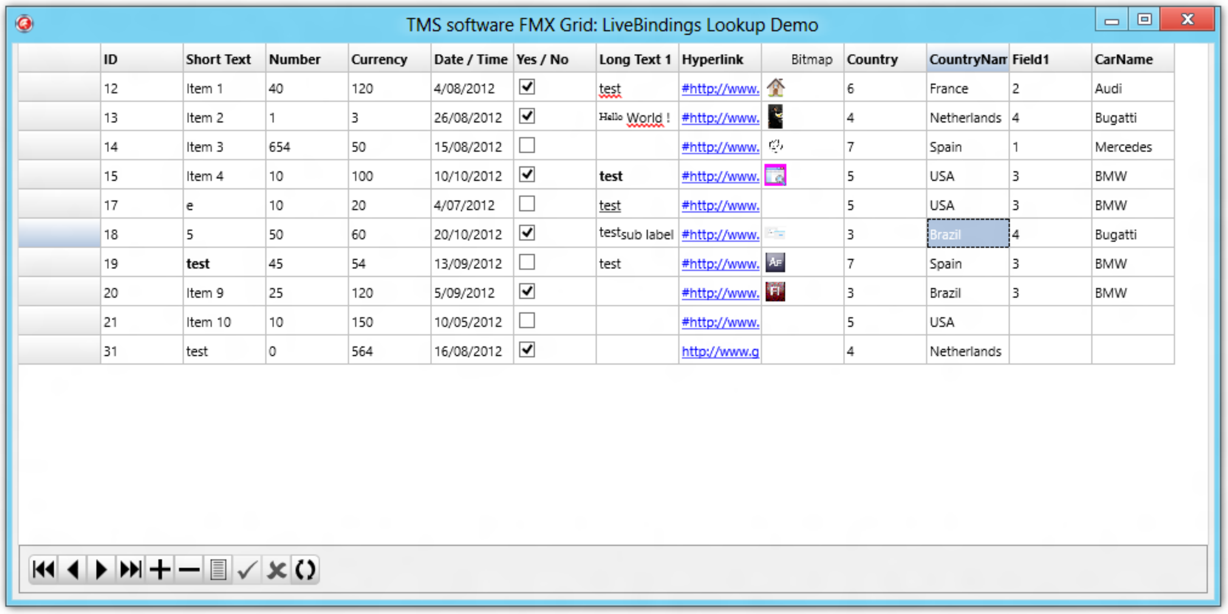
When editing, some editors are numeric only, the datetime shows a calendar and the checkbox is used on a boolean field. These type of cells and editors are automatically detected and used by the grid.

Based on the type of column, the editor changes. There is support for alpha- and numeric editing, boolean and datetime editing. For that last type the TDateEdit / TDatePicker component is used. If the automatically chosen editor is not suitable for your application you can override the OnGetCellEditorType event that allows specifying a different editor.

Deleting and inserting a row

The delete key deletes a record with a confirmation dialog. The insert key inserts a new row. This can be switched off in grid.options.keyboard.

LiveBindings Lookup demo


TMS software FMX Grid: LiveBindings Lookup Demo

	ID	Short Text	Number	Currency	Date / Time	Yes / No	Long Text 1	Hyperlink	Bitmap	Country	CountryNam	Field1	CarName
	12	Item 1	40	120	4/08/2012	☑	test	#http://www.		6	France	2	Audi
	13	Item 2	1	3	26/08/2012	☑	Hallo World !	#http://www.		4	Netherlands	4	Bugatti
	14	Item 3	654	50	15/08/2012	☐		#http://www.		7	Spain	1	Mercedes
	15	Item 4	10	100	10/10/2012	☑	**test**	#http://www.		5	USA	3	BMW
	17	e	10	20	4/07/2012	☐	test	#http://www.		5	USA	3	BMW
	18	5	50	60	20/10/2012	☑	testsub label	#http://www.		3	Brazil	4	Bugatti
	19	**test**	45	54	13/09/2012	☐	test	#http://www.		7	Spain	3	BMW
	20	Item 9	25	120	5/09/2012	☑		#http://www.		3	Brazil	3	BMW
	21	Item 10	10	150	10/05/2012	☐		#http://www.		5	USA		
	31	test	0	564	16/08/2012	☑		http://www.g		4	Netherlands		

Shows how to bind multiple lookup fields to a combobox in the grid used as an inplace editor. More information can be found in the LiveBindings chapter.