TMS FNC Grid
DEVELOPERS GUIDE

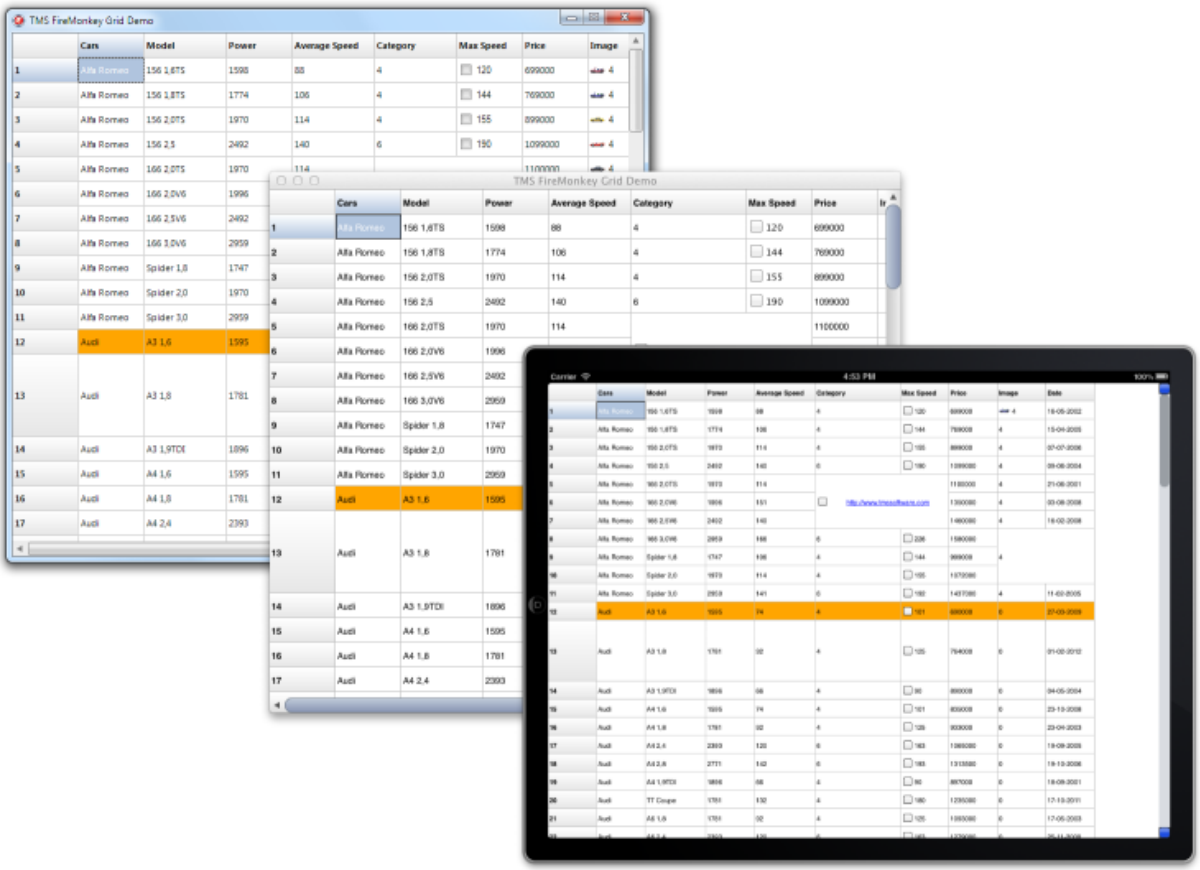
June 2018

Web: http://www.tmssoftware.com
Email: info@tmssoftware.com

Index

Introduction

The TMS FNC Grid offers a fully cross-platform, high-performing, versatile and feature packed grid. It is built from the ground up and at the same time, it is sufficiently similar to the VCL TAdvStringGrid to make developers used to TAdvStringGrid quickly familiar and up & running.



Grid properties

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

Columns: TTMSFNCGridColumns: 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: TTMSFNCGridScrollMode: 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.

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: TTMSFNCGridBorders: Sets which borders (vertical, horizontal or all) are visible on normal cells.
FixedCellBorders: TTMSFNCGridBorders: 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: TTMSFNCGridClipboardPasteAction: 2 options: overwrites the data of the cells within range of the clipboard data, or inserts new rows and columns from the focused cell

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.
DirectComboDrop: Boolean: Enables direct combo drop when clicking on a cell and the editor is set to a combo type editor.

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:TTMSFNCGridFilterRows: 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 Options.Selection.Mode 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.

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 & , " 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   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.

Selection

Mode: TTMSFNCGridSelectionMode: 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.

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: TTMSFNCGridDeleteKeyHandling 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: TTMSFNCGridEnterKeyHandling: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: TTMSFNCGridInsertKeyHandling: 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: TTMSFNCGridTabKeyDirection: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: TTMSFNCGridTabKeyHandling: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.
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 selection mode.
TouchScrollingSensitivity: single: The sensitivity of the touch scrolling.
WheelScrollSize: integer: The amount of cells that are scrolled when using the mouse-wheel to navigate.

Sorting

BlankPosition: TTMSFNCGridSortBlankPosition: 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: TTMSFNCGridSortColumns: 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: TTMSFNCGridSortingMode: 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: TTMSFNCGraphicsColor: 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 := MakeCellRange(StartCol,StartRow,EndCol,EndRow);
grid.FocusedCell := MakeCell(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.Options.Selection.Mode: TTMSFNCGridSelectionMode;
```

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

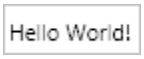
The keyboard interaction can be modified with the grid.Options.Keyboard.InsertKeyHandling and grid.Options.Keyboard.DeleteKeyHandling properties.

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.

Cell Properties

Cells[Col,Row: Integer]: string

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



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

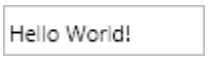
Returns the text of a cell with all HTML formatting removed.

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.

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.



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]: TTMSFNCGraphicsColor

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.TMSFNCGrid1GetCellLayout(Sender: TObject; ACol, ARow:
Integer;
  ALayout: TTMSFNCGridCellLayout; ACellState: TCellState);
begin
  if (ACol = 3) and (ARow >= TMSFNCGrid1.FixedRows) then
    ALayout.Fill.Color := gcRed;
end;
```

Angles[Col,Row: Integer]: single

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

HorzAlignments[Col,Row: Integer]: TTMSFNCGraphicsTextAlign

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.TMSFNCGrid1GetCellLayout(Sender: TObject; ACol, ARow:
Integer;
  ALayout: TTMSFNCGridCellLayout; ACellState: TCellState);
begin
```

```
  if ARow < TMSFNCGrid1.FixedRows then
    ALayout.TextAlign := TTMSFNCGraphicsTextAlign.taCenter;
end;
```

VertAlignments[Col,Row: Integer]: TTMSFNCGraphicsTextAlign

Changes the vertical text alignment in a cell.

FontSizes[Col,Row: Integer]: single

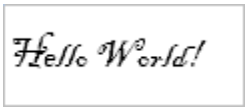
Changes the size of the font of the text in a cell.

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]: TTMSFNCGraphicsColor

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.TMSFNCGrid1GetCellLayout(Sender: TObject; ACol, ARow:
Integer;
  ALayout: TTMSFNCGridCellLayout; ACellState: TCellState);
var
  cv: Double;
begin
  if (ACol >= TMSFNCGrid1.FixedColumns) and (ARow >= TMSFNCGrid1.FixedRows)
then
  begin
    cv := TMSFNCGrid1.AllFloats[ACol,ARow];
    if cv < 0 then
      ALayout.Font.Color := gcRed
    else
      ALayout.Font.Color := gcGreen;
  end;
end;
```

ReadOnlys[Col,Row: Integer]: boolean

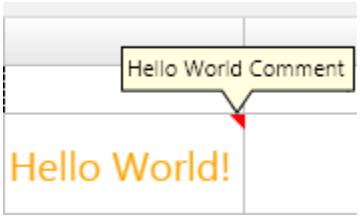
Sets a cell readonly, which means that the cell is no longer editable via the user-interface. The equivalent event for dynamically controlling this is OnGetCellReadOnly.

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]: TTMSFNCGraphicsColor

Sets the color of the comment triangle.


Hello World!

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.TMSFNCGrid1GetCellClass(Sender: TObject; ACol,
  ARow: Integer; var CellClassType: TTMSFNCGridCellClass);
begin
  if (ACol = 4) and (ARow = 3) then
    CellClassType := TTMSFNCBitmapGridCell;
end;
```

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.

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

```
procedure TForm1.TMSFNCGrid1GetCellLayout(Sender: TObject;
  ACol, ARow: Integer; ACellState: TCellState);
begin
  case ACellState of
    csNormal: ALayout.Fill.Color := gcRed;
    csFocused: ALayout.Fill.Color := gcBlue;
    csFixed: ALayout.Fill.Color := gcGreen;
    csFixedSelected: ALayout.Fill.Color := gcOrange;
    csSelected: ALayout.Fill.Color := gcLime;
  end;
end;
```

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.TMSFNCGrid1CanInsertColumn(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.TMSFNCGrid1CanAppendRow(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.TMSFNCGrid1CanAppendColumn(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.TMSFNCGrid1CanDeleteRow(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.TMSFNCGrid1GetCellEditorType(Sender: TObject;
  ACol, ARow: Integer; var CellEditorType: TTMSFNCGridEditorType);
begin
  if (ACol = 4) and (ARow = 2) then
    CellEditorType := etCustom;
end;

procedure TForm1.TMSFNCGrid1GetCellEditorCustomClassType(
  Sender: TObject; ACol, ARow: Integer;
  var CellEditorCustomClassType: TTMSFNCGridEditorClass);
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.TMSFNCGrid1CellEditGetData(Sender: TObject; ACol,
  ARow: Integer; CellEditor: TTMSFNCGridEditor; 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]: TTMSFNCGraphicsColor;

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: TTMSFNCEdit class type.
et*EditBtn: TTMSFNCEditBtn 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.

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

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.TMSFNCGrid1CellBeforeDraw(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 TTMSFNCBitmapGridCell or the cell has been added with grid.AddBitmap or grid.AddBitmapName procedures.

OnCellRadioButtonClick: Event called when clicking on the radiobutton of a cell. The return type for the OnGetCellClass is TTMSFNCRadioGridCell 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 TTMSFNCFixedGridCell or TTMSFNCCommentGridCell and the cell has filtering enabled in case of TTMSFNCFixedGridCell or has set a comment in case of TTMSFNCCommentGridCell and the popup is shown by clicking the dropdown button or the comment triangle.

OnCellCheckBoxClick: Event called when clicking on the checkbox of a cell. The return type for the OnGetCellClass is TTMSFNCCheckGridCell 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 TTMSFNCCommentGridCell 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 TTMSFNCFixedGridCell, 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 TTMSFNCNodeGridCell 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.

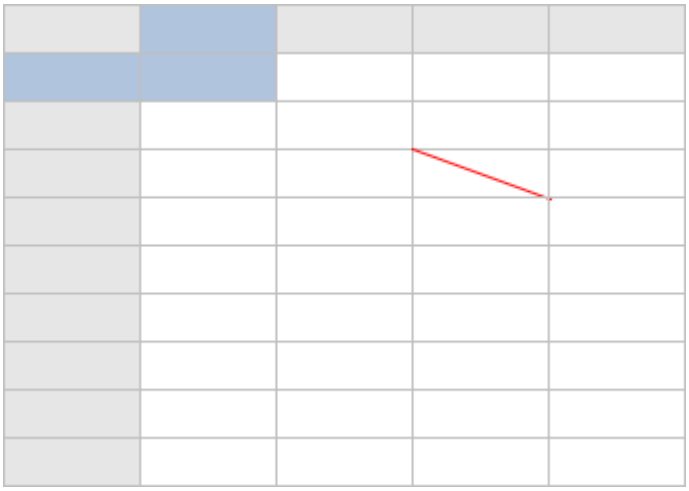
OnCellClick: Event called when a cell is clicked.

OnFixedCellClick: Event called when a fixed cell is clicked.

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 TForm135.TMSFNCGrid1CellAfterDraw(Sender: TObject; ACol,
  ARow: Integer; AGraphics: TTMSFNCGraphics; ARect, ATextRect:
TRectF);
begin
  if (ACol = 3) and (ARow = 3) then
  begin
    AGraphics.SetStrokeColor(gcRed);
    AGraphics.DrawLine(PointF(ARect.Left, ARect.Top),
PointF(ARect.Right, ARect.Bottom));
  end;
end;
```



Custom Cell Class

When dropping a default grid on the form, you will notice fixed and normal cells. The basic implementation supports a fill, stroke and a text. For a normal default cell, the cell class type is TTMSFNCGridCell. 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 TTMSFNCFixedGridCell 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. 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: TTMSFNCGridCell to the type specified for the cell.

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

```
procedure TForm4.TMSFNCGrid1GetCellClass(Sender: TObject; ACol, ARow:
Integer;
  var CellClassType: TTMSFNCGridCellClass);
begin
  if (ARow >= TMSFNCGrid1.FixedRows) and (ACol = 3) then
    CellClassType := TTMSFNCCheckGridCell;
end;
```

The equivalent code to programmatically add checkboxes is:

```
var
  i: integer;
begin
  for i := TMSFNCGrid1.FixedRows to TMSFNCGrid1.RowCount - 1 do
    TMSFNCGrid1.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.TMSFNCGrid1GetCellProperties(Sender: TObject; ACol,
  ARow: Integer; Cell: TTMSFNCGridCell);
begin
  if (ACol = 3) and (ARow >= TMSFNCGrid1.FixedRows) and (Cell is
TTMSFNCCheckGridCell) then
  begin
    (Cell as TTMSFNCCheckGridCell).CheckBox.IsChecked := true;
```

```
  end;
end;
```

TTMSFNCGridCell

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

TTMSFNCRadioGridCell

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

The methods to add & remove radiobuttons are:

TMSFNCGrid.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.

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

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

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

TMSFNCGrid.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:

```
TMSFNCGrid1.AddRadioButton(1, 1, 4, True);
TMSFNCGrid1.AddRadioButtonColumn(1, 1);
```

TTMSFNCCheckGridCell

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

The methods to add & remove checkboxes are:

TMSFNCGrid.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.

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

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

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

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

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

TMSFNCGrid.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 TMSFNCGrid.CheckTrue, it will be displayed as checked. When the text value of the cell equals TMSFNCGrid.CheckFalse, it will be displayed as unchecked. To get or set the checkbox state of this checkbox type, use:

TMSFNCGrid.Cells[Col,Row] := TMSFNCGrid.CheckTrue.

or

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

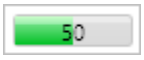
TMSFNCGrid.AddDataCheckBoxColumn(Col: integer);
Adds data checkboxes in all cells of column Col.

Examples:

```
TMSFNCGrid1.AddCheckBox(1, 1, True);
TMSFNCGrid1.AddCheckBoxColumn(1);
```

TTMSFNCProgressGridCell

Inherits from TTMSFNCGridCell 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:

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

TMSFNCGrid.AddButton(Col,Row: Integer; AText: string; AWidth: Integer = 20; AHeight: Integer = 20);
Adds a button with a specific text, width and height. Generates the OnCellButtonClick event when clicked.

TMSFNCGrid.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].

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

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

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

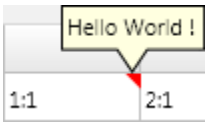
TMSFNCGrid.RemoveProgressBar(Col,Row: Integer);
Removes the progressbar from cell Col,Row.

Examples:

```
TMSFNCGrid1.AddProgressBar(1, 1, 50);
```

TTMSFNCCommentGridCell

Inherits from TTMSFNCGridCell 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:

TMSFNCGrid.Comments[Col,Row]: string;
TMSFNCGrid.CommentColors[Col,Row]: TTMSFNCGraphicsColor;

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

Examples:

```
TMSFNCGrid1.Comments[1, 1] := 'Hello World!';
TMSFNCGrid1.CommentColors[1, 1] := gcRed;
TMSFNCGrid1.Comments[2, 2] := ''; // remove comment from cell 2,2
```

TTMSFNCFixedGridCell

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

TTMSFNCNodeGridCell

Inherits from TTMSFNCGridCell 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:

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

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

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

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

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

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

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

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

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

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

TMSFNCGrid.OpenAllNodes;
Opens all nodes in the grid.

TMSFNCGrid.CloseAllNodes;
Closes all nodes in the grid.

TTMSFNCBitmapGridCell

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

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

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

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

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

TMSFNCGrid.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.

TMSFNCGrid.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

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

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

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

TMSFNCGrid.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()

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

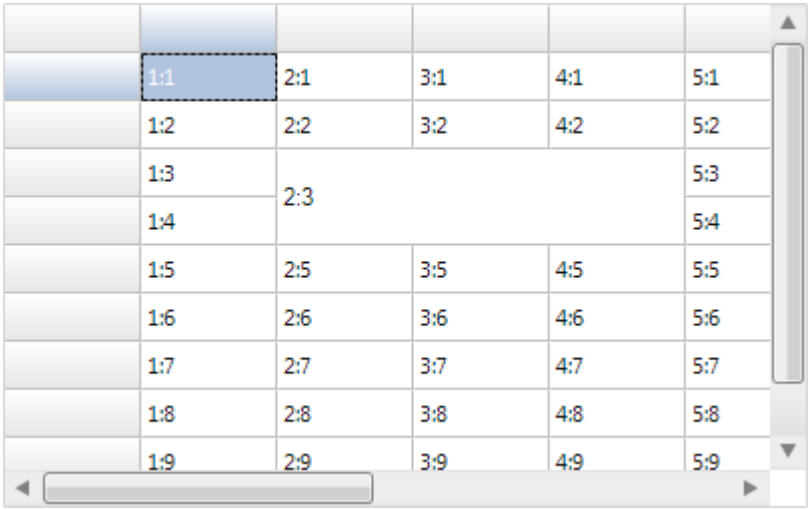
Grid cell merging / splitting

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

Sample:

```
TMSFNCGrid1.MergeCells(2, 3, 3, 2);
TMSFNCGrid1.MergeSelection(TMSFNCGrid1.CellRange(2, 3, 3, 2));
```



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.

Sample:

```
TMSFNCGrid1.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.

Editing

By default the grid supports editing, this can be turned on and off with grid.Options.Editing.Enabled. The default editor is a TTMSFNCEdit 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:

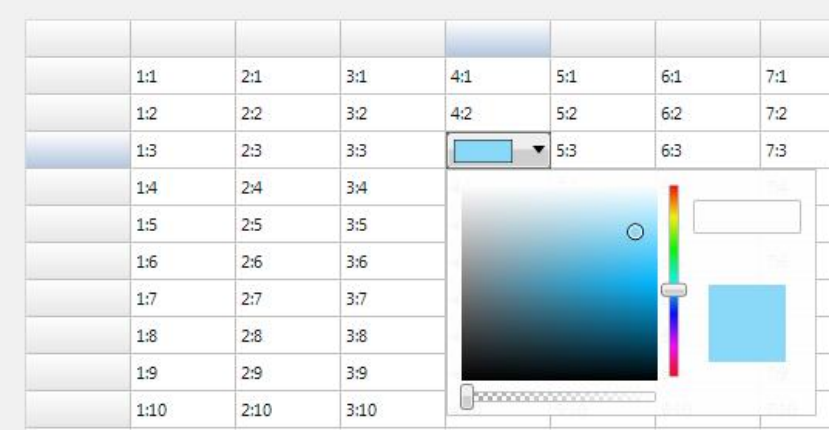
Under grid.Options.Keyboard, grid.Options.Mouse, grid.Options.Editing there are various properties that can be used to customize the way the editing occurs, from direct editing to navigating from edit to edit with the arrow keys, as well as enterkeyhandling to jump to the next row or column, … .

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

etEdit, etNumericEdit, etSignedNumericEdit, etFloatEdit, etSignedFloatEdit, etUppercaseEdit, etMixedCaseEdit, etLowerCaseEdit, etMoneyEdit, etHexEdit, etAlphaNumericEdit, etValidCharsEdit, etComboBox, etSpinBox, etDatePicker, etColorPicker, etTrackBar, etMemo, etCustom.

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

```
procedure TForm1.TMSFNCGrid1GetCellEditorType(Sender: TObject; ACol,
  ARow: Integer; var CellEditorType: TTMSFNCGridEditorType);
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:

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:

```
TMSFNCGrid1.ColumnWidths[4] := 150;
TMSFNCGrid1.RowHeights[4] := 100;

procedure TForm1.TMSFNCGrid1GetCellEditorType(Sender: TObject; ACol,
  ARow: Integer; var CellEditorType: TTMSFNCGridEditorType);
begin
  CellEditorType := etCustom;
end;
```

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

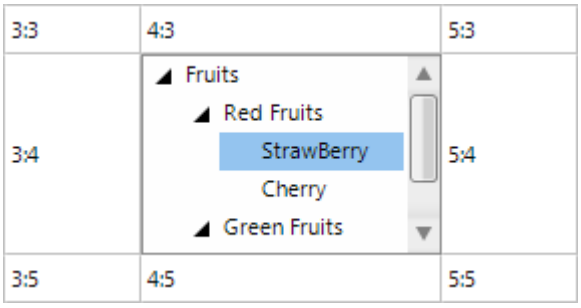
```
procedure TForm1.TMSFNCGrid1GetCellEditorCustomClassType(Sender:
TObject;
  ACol, ARow: Integer; var CellEditorCustomClassType:
TTMSFNCGridEditorClass);
begin
  CellEditorCustomClassType := TTreeView;
end;
```

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

```
procedure TForm1.TMSFNCGrid1GetCellEditorProperties(Sender: TObject;
ACol,
```

```
  ARow: Integer; CellEditor: TTMSFNCGridEditor);
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.

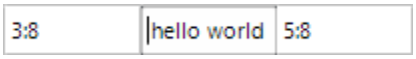
```
procedure TForm1.TMSFNCGrid1CellEditDone(Sender: TObject; ACol, ARow:
Integer;
  CellEditor: TTMSFNCGridEditor);
```

```
begin
  TMSFNCGrid1.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.TMSFNCGrid1CellEditGetData(Sender: TObject; ACol,
  ARow: Integer; CellEditor: TTMSFNCGridEditor; 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 TTMSFNCGrid.Options.Selection.Mode. 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:

```
TMSFNCGrid1.Selection := MakeCellRange(2,2,4,4);
```

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

```
TMSFNCGrid1.Selection := MakeCellRange(3,3,3,3);
```

When the Options.Selection.Mode 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 TMSFNCGrid1.RowCount - 1 do
  begin
    if TMSFNCGrid1.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 TMSFNCGrid1.RowSelectionCount - 1 do
  begin
    rowindex := TMSFNCGrid1.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 TMSFNCGrid1.CellSelectionCount - 1 do
```

```
  begin
    c := TMSFNCGrid1.SelectedCell[i];
    // do something with the selected cell
    TMSFNCGrid1.Cells[c.Col, c.Row] := TMSFNCGrid1.Cells[c.Col, c.Row] + '*';
  end;
end;
```

In combination with the Options.Selection.Mode, 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:

TMSFNCGrid.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.

TMSFNCGrid.ColumnAvg(ACol: integer; FromRow: integer = -1; ToRow: Integer = -1): Double;
Calculates the average cell value of cells within a column.
TMSFNCGrid.ColumnMin(ACol: integer; FromRow: integer = -1; ToRow: Integer = -1): Double;
Calculates the minimum cell value of cells within a column.
TMSFNCGrid.ColumnMax(ACol: integer; FromRow: integer = -1; ToRow: Integer = -1): Double;
Calculates the maximum cell value of cells within a column.
TMSFNCGrid.ColumnDistinct(ACol: integer; FromRow: integer = -1; ToRow: Integer = -1): Double;
Counts the number of distinct cell valuess within a column.
TMSFNCGrid.ColumnStdDev(ACol: integer; FromRow: integer = -1; ToRow: Integer = -1): Double;
Calculates the number of standard deviation of cell values within a column.
TMSFNCGrid.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:

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

Example:

The grid is initialized with:

```
begin
  TMSFNCGrid1.RowCount := 20;
  TMSFNCGrid1.FixedFooterRows := 1;
  TMSFNCGrid1.RandomFill(false,100);
  TMSFNCGrid1.ColumnCalculation[1] := ccSUM;
  TMSFNCGrid1.ColumnCalculation[2] := ccAVG;
  TMSFNCGrid1.ColumnCalculation[3] := ccMIN;
  TMSFNCGrid1.ColumnCalculation[4] := ccMAX;
  TMSFNCGrid1.UpdateCalculations;
end;
```

and shows upon starting the application:

	27	10	2	50
	13	72	91	51
	12	21	42	95
	87	57	81	55
	23	79	58	65
	90	31	94	28
	38	73	33	21
	40	21	22	90
	40	28	10	93
	887	**49.8888888**	**2**	**95**

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 (Windows and macOS only)
XML: Saves the grid cell data to XML file
PDF: Saves the grid cell data to PDF 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

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

```
TMSFNCGrid.IOOffset: TPoint
```

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

```
TMSFNCGrid.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.

PDF Files

Drop a TTMSFNCGridPDFIO component on the form and connect the TTMSFNCGrid to this non-visual component's Grid property.

Export

Simply call:

```
TMSFNCGridPDFIO.Save(FileName);
```

Alternatively the TMSFNCGridPDFIO component is able to save to a stream. Simply call the Save method with a TStream instance.

Additionally, the TMSFNCGridPDFIO component is capable of configuring the margins, header, footer as well as PDF meta-data such as the creator, author title and keywords. These properties are found under Options and Information.

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 (Windows and macOS only)

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

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

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

  ```
  TMSFNCGridExcelIO.XLSImport(FileName);
  ```

  or

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

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

  ```
  TMSFNCGridExcelIO.XLSExport(Filename);
  ```

  or

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

Properties of TTMSFNCGridExcelIO

Many properties are available in TTMSFNCGridExcelIO 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

Advanced topics on exporting & importing

To apply transformations on cell data for loading and saving it is easy to create a descendent class from TTMSFNCGrid 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(TTMSFNCGrid)
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.

		▲		
	34	2	67	40
	97	15	44	48
	43	18	86	23
	0	43	86	20
	86	45	97	87
	88	47	47	13
	60	57	31	50

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

		▼		
	30	85	60	66
	64	77	63	61
	60	57	31	50
	88	47	47	13
	86	45	97	87
	0	43	86	20
	43	18	86	23

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:

```
TTMSFNCGridSortFormatEvent = 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 TTMSFNCGridOnRawCompare(Sender: TObject; ACol, Row1, Row2:
Integer; var Res: Integer);
var
  c1,c2: Integer;
begin
  c1 := integer(TMSFNCGrid1.Objects[ACol,Row1]);
  c2 := integer(TMSFNCGrid1.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.

```
  TMSFNCGrid1.SortIndexes.Clear;
  TMSFNCGrid1.SortIndexes.AddIndex(3, sdAscending);
  TMSFNCGrid1.SortIndexes.AddIndex(4, sdDescending);
  TMSFNCGrid1.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.

		1	2	3
	9	1	39	8
	34	10	55	67
	3	20	6	72
	33	40	49	50
	84	56	26	49

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 TMSFNCGrid.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 TMSFNCGrid.SortGroupedIndex;

```
TMSFNCGrid1.SortIndexes.Clear;
TMSFNCGrid1.SortIndexes.AddIndex(5, sdDescending);
TMSFNCGrid1.SortIndexes.AddIndex(2, sdAscending);
TMSFNCGrid1.SortGroupedIndexed;
```

Grouping

TTMSFNCGrid 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
TMSFNCGrid1.LoadFromCSV('cars.csv');
TMSFNCGrid1.ColWidths[0] := 20;

// do grouping on column 1
TMSFNCGrid1.Group(1);

// apply grouped sorting on (new) column 1
TMSFNCGrid1.SortGrouped(1, sdAscending);
```

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

```
TMSFNCGrid.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

TTMSFNCGrid 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.TMSFNCGrid1GroupCalc(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 + TMSFNCGrid1.Floats[ACol,i];
end;
m := m / (ToRow - FromRow + 1);

// calculate standard deviation
sd := 0;
for i := FromRow to ToRow do
begin
  sd := sd + sqr(TMSFNCGrid1.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.

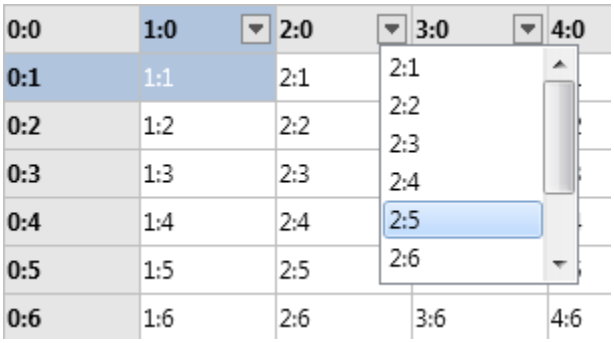
BorderColor: TTMSFNCGraphicsColor: Border color of a column grid cell in normal state.
BorderWidth: Single: Border width of a column grid cell in normal state.
Color: TTMSFNCGraphicsColor: Color of a column grid cell in normal state.
ColumnType: TTMSFNCGridColumnType: Identifies the type of the column. 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: TTMSFNCGridEditorType: 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: TTMSFNCGraphicsColor: 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: TTMSFNCGraphicsTextAlign: 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: TTMSFNCGraphicsTextAlign: 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.

Filtering

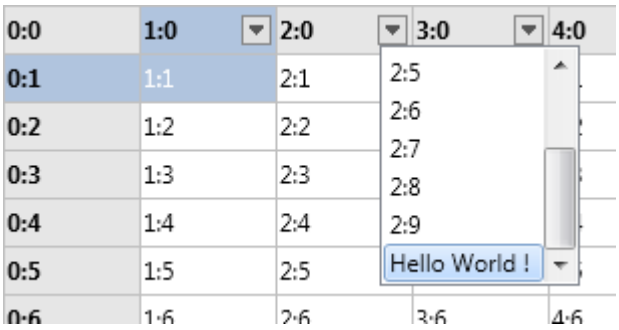
The TTMSFNCGrid also supports built-in filtering. Filtering can be done programmatically or via the user interface when enabling the dropdown button on a fixed column header cell. When the dropdown button is visible, the dropdown list is automatically filled with unique values from the column. When selecting an item from the dropdown list, the grid is filtered based on the value you have selected:

```
TMSFNCGrid1.Options.Filtering.DropDown := True;
TMSFNCGrid1.LinearFill(True);
```



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.TMSFNCGrid1NeedFilterDropDownData(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.TMSFNCGrid1NeedFilterDropDownData(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.TMSFNCGrid1FilterSelect(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 TMSFNCGrid1.Filter.Add do
begin
  Condition := '2:4';
  Column := 2;
end;
```

TMSFNCGrid.ApplyFilter;

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

The TTMSFNCGridFilterData 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: TTMSFNCGridFilterData;
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: TTMSFNCGridFilterData;

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;
```

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 gcRed,gcYellow,gcWhite ... 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="gcRed">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="gcYellow">This has a yellow background</P>
Example : <P align="right" bgcolor="gcYellow" bgcolorto="gcRed">This has a gradient background</P>*

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

BR : linebreak

 : 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="gcYellow"> : sets background color to yellow
<BODY background="file://c:\test.bmp"> : sets tiled background to file test.bmp
<BODY bgcolor="gcYellow" bgcolorto="gcWhite" 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 ⁹/₁₆ 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

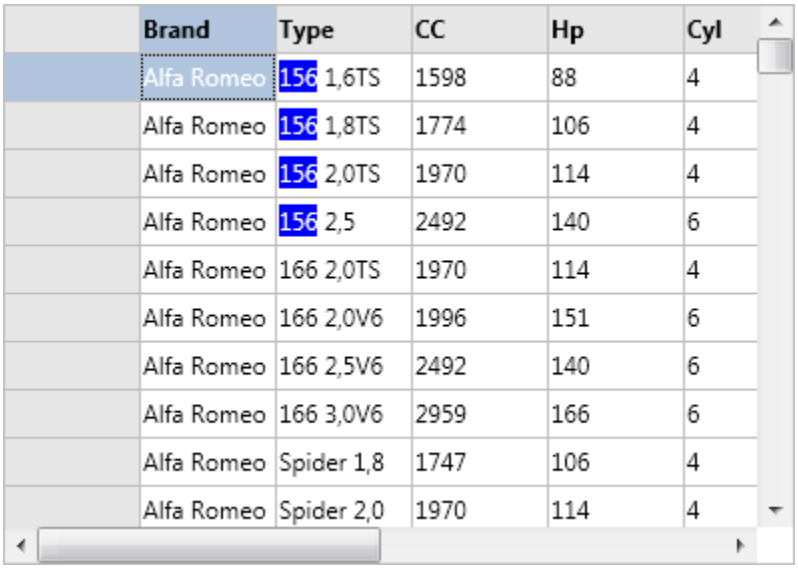
SHAD : text with shadow
<SHAD> : start text with shadow
</SHAD> : end text with shadow

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

Special characters
Following standard HTML special characters are supported :
< : less than : <
> : greater than : >
& : &
" : "
  : non breaking space
™ : trademark symbol
€ : euro symbol
§ : section symbol
© : copyright symbol
¶ : paragraph symbol

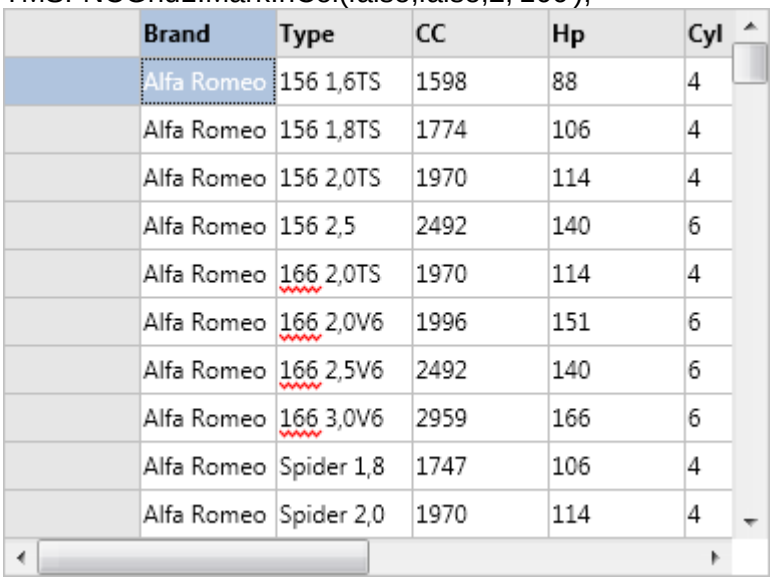
When hyperlinks are specified in grid cells, these hyperlinks are displayed underlined and in blue color. When the hyperlink is clicked, the OnCellAnchorClick event is triggered. Via HTML formatting, the grid also offers highlighting or marking of text in grid cells. This can be used to indicate text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

```
TMSFNCGrid1.HighlightInCol(false,false,2,'156');
```



	Brand	Type	CC	Hp	Cyl
	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
	Alfa Romeo	Spider 2,0	1970	114	4

TMSFNCGrid1.MarkInCol(false,false,2,'166');



	Brand	Type	CC	Hp	Cyl
	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
	Alfa Romeo	Spider 2,0	1970	114	4

Available methods:

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

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

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

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

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

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

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

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

TMSFNCGrid.UnHighlightAll;
Remove the highlighting in all cells.

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

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

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

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

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

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

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

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

Databinding

The TTMSFNCGrid supports databinding via the TTMSFNCGridDatabaseAdapter. This adapter is responsible for connecting to the dataset and for loading, displaying and editing the data. The adapter exposes a set of properties for data visualization and manipulation and a column collection that represents the available fields. Below is an overview of the available properties and events.

Active: Activates the adapter and displays the data from the dataset when the dataset is active.
AutoCreateColumns: When true, automatically retrieves the fields from the dataset and creates columns for each field. The columns in the grid match the columns in the dataset.
AutoRemoveColumns: When true, automatically removes all columns when the dataset deactivates or changes.
Columns: A column collection which is automatically filled with data from the dataset when AutoCreateColumns is true. The column collection contains properties to change the visualization and manipulation of data for each column. The columns collection on adapter level is automatically synchronized with the columns on grid level and work together.
DataSetType: The type of dataset (sequential or non-sequential). Automatically preset when DataSetTypeAuto is set to true. The DataSetType property can be set manually when DataSetTypeAuto is set to false.
DataSetTypeAuto: When true, automatically determines the type of dataset (sequential or non-sequential) and presets the DataSetType property.
DataSource: The datasource, connected to the dataset.
Grid: The grid, to display the data from the dataset.
PostMode: The post mode when editing an existing or new record. By default, the PostMode is set to apmRow which means that the new information is posted when the row changes. The alternative is apmCell, which means that the new information is posted when the cell changes. To manually handle data posting, set the PostMode to apmNone.
ShowBooleanFields: When true, shows a checkbox and tries to convert the value to a Boolean when Boolean information is available.
ShowMemoFields: When true, the grid will display the contents of the memo field instead of displaying a placeholder.
ShowPictureFields: When true, the grid will display the image from a blob field or graphic field.

At column level, the following properties are available:

CheckBoxField: When true, the field is treated as a Boolean field represented by a checkbox.
CheckFalse: Sets the value of the DB field that corresponds with an unchecked checkbox.
CheckTrue: Set sthe value of the DB field that corresponds with a checked checkbox.
FieldName: Sets the fieldname for the dataset field that is displayed in the column.

Header: Sets the column header text. When empty, the FieldName.DisplayLabel is used as a column header text.
HTMLTemplate: Sets the HTML template of the column. This allows to display multiple DB fields in one column.
PictureField: When true and a blob field is connected to the column, the blob data is treated as an image and displayed.
ProgressField: When true, the field value in the column is treated as a value to set the progressbar. Note that this requires that the field value is between 0 and 100. If this is not available in the dataset, a calculated field could be used to generate a value between 0 and 100.
UseColumnEditor: Specifies whether to use the editor defined at grid column level or the one automatically determined by the database adapter.
UseLookupEditor: Specifies whether to use a lookup editor combobox for the column.

Memo fields, Boolean fields & image blobs

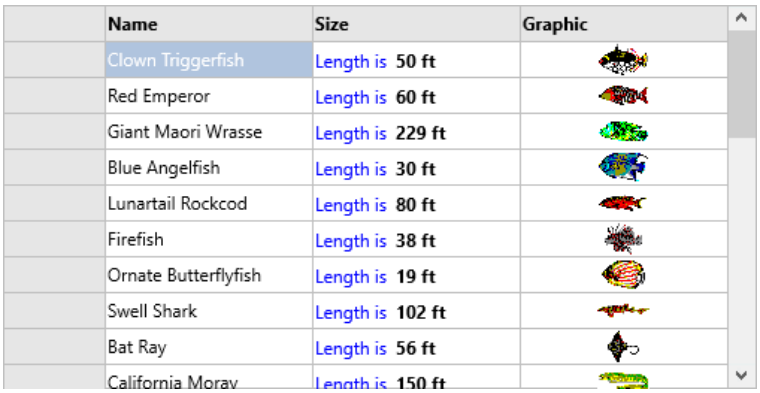
The TTMSFNCGridDatabaseAdapter can automatically display memo fields, can show Boolean fields as checkboxes in the grid and can display images stored in blob fields. By default, a memo field is displayed as '(MEMO)' and an image blob is displayed as '(GRAPHIC)' just like in the default TGrid or TStringGrid in FMX, VCL and LCL.
To show real contents of memo fields, set global property ShowMemoFields = true. To have the cell sizes automatically sized according to the text in memo fields, it is sufficient to call Grid.AutoSizeRows(false,4) after activating the dataset. By default, a DB field of the type ftBoolean is displayed as value 'true' or 'false".

The adapter can also automatically show such ftBoolean field type as a checkbox. To do this, set ShowBooleanFields = true. One step further is that the adapter can also show checkboxes for DB fields that do not have the type ftBoolean but that can have two values. This can be used with databases that do not support Boolean field types for example where a true condition is stored in a field as one value and false condition is stored as another value. To show checkboxes for such fields, set Columns[columnindex].CheckBoxField = true and set via the properties Columns[columnindex].CheckTrue, DBAdvGrid.Columns[columnindex].CheckFalse the values for a true condition and false condition.

The adapter can also show BMP, GIF, JPEG images that are stored in BLOB fields. By default, such fields are displayed as '(GRAPHIC)'. When setting ShowPictureFields = true, all fields with the type ftGraphic will be displayed as images. In some databases, the field type for an image field will not be ftGraphic but just ftBlob for example. The grid cannot automatically know that the BLOB data should be interpreted as an image though. If a DB field of the type ftBlob holds images, Columns[columnindex].PictureField can be set to true and for this column, the grid will try to display the BLOB data as BMP, GIF or JPEG image.

HTML Templates

Through a HTML template, it is possible to put multiple fields in a single cell with optionally HTML formatting. This is done by specifying a HTML template via the property Columns[Index].HTMLTemplate. The HTML template is a string where a DB field reference set by <#DBFIELDNAME> will be replaced by the DB field value for each record for display. For example, when a table has a field Length (cm), the following HTML template creates a single cell with text in blue & bold: Columns[2].HTMLTemplate := '<FONT color="gcBlue">Length is </FONT> <B><#Length (cm)> ft</B>'; The resulting grid looks like:



Editing

The adapter supports various inplace editors as well as the capability to use external controls as inplace editor for the grid. The "Editing" chapter in this document has an overview of all possible inplace editors that are included and how to use external editors as well.
To enable editing in the grid, it is required that

- The database table is editable
- The database field is editable

The adapter can perform editing in two modes. This is selected by the property PostMode. When this is set to apmCell, the value that has been edited is posted immediately to the database when a cell leaves inplace editing. When PostMode is set to apmRow, a post operation will only occur when all cells in a row have been edited and the user moves to another row. Typically, for tables with various required fields, apmRow is the preferred setting.

When a selected cell is clicked, the inplace editing starts. It is important to know that editing of a cell or row stops when the inplace editor loses focus. Only when it loses focus, the edited value is either directly posted (apmCell mode) or internally stored (apmRow mode) to post when the row changes.