What's new in EhLib.VCL 10.0

(New features of your applications)

Brief description of new features in this version:

* The WriteDataSetToMemTable method has been added to the TDatasetFeaturesEh class for fast data loading from different DataSets into TMemTableEh.
* Added the ability to load Xlsx files into the TXlsMemFileEh class.
* The capabilities of the TXlsMemFileEh class have been expanded in terms of data management: Grouping of rows, Parameters of rows (Height, Visibility), Scale, Printing parameters, etc.
* Extended version of procedures for unloading data from DBGridEh to Xlsx file.
* Extended version of procedures for unloading data from DBGridEh to text file.
* Search history in DBGridEh.SeachPanel
* High DPI monitors support
* Changes in the interface of the library modules
* Working with the TXlsMemFileEh class

# ***The WriteDataSetToMemTable method has been added to the TDatasetFeaturesEh class to fast data loading from different DataSets into TMemTableEh.***

The TMemTableEh.LoadFromDataSet method now has the ability to “quickly” load data without using the classic version of DataSet.Append, DataSet.Fields[i].Value: = v.

Fast loading will be used only for DataSets for which the TDatasetFeaturesEh class is registered with the overloaded function WriteDataSetToMemTable.

In the current version of the library, fast loading works for DataSets such as TADOQuery and TCustomADODataSet and its heirs.

For an example of implementation of the WriteDataSetToMemTable method, see the TADOSQLDatasetFeaturesEh class (EhLibADO Module). TADOSQLDatasetFeaturesEh.WriteDataSetToMemTable method.

# ***Added the ability to load Xlsx files into the TXlsMemFileEh class.***

The LoadFromFile method has been added to the TXlsMemFileEh class.

Now it is possible to load the contents of the Xlsx file into the TXlsMemFileEh object, change the contents of the table and save it back to the file.

Most (but not all) Xlsx file elements are supported when loading data.

In the current version of the library, the LoadFromFile and XlsMemFile functions support the following Xlsx file elements.

* WorkSheet - worksheets. Worksheet names and colors.
* Worksheet Rows and Columns. Width\Height, visibility of rows and columns.
* The contents of the cells in the worksheet. Text, numbers, dates.
* Cell fill color and pattern.
* Color, title, size and style of the font.
* Alignment, text wrapping, merging of cells.
* Vertical, horizontal, oblique dividing lines.
* Print properties: Scale, sheet titles.
* Autofilter.
* Fixed-non-scrollable areas of the sheet.
* Formulas (only as text, no calculation).
* Grouping Rows, Grouping Columns.

The current version does not support:

Gradient filling of cells

Formatting individual sections of text within a cell (RichText in a cell)

Conditional Formatting (Formula-Based Formatting)

See an example of using loading and modifying Xlsx files in Demo projects:

<EhLib Archive>\Demos\XlsFileEh\Project1.dpr

<EhLib Archive>\Demos\DBGridEh.ExportToXlsx\Project1.dpr

<EhLib Archive>\Demos\MainDemo\Project1.dpr (Toolbar button - Export to Excel).

# ***The capabilities of the TXlsMemFileEh class have been expanded in terms of data management: Grouping of rows, Parameters of rows (Height, Visibility), Scale, Printing parameters, etc.***

The capabilities of the TXlsMemFileEh class in terms of data management have been expanded: Grouping of rows, Parameters of rows (Height, Visibility), Scale, Column width by default, Height of a row by default, Printing options, Color of the worksheet bookmark, Page indents.

# ***Extended version of procedures for unloading data from DBGridEh to Xlsx file.***

In version 9.5 there was a function for unloading DBGridEh into an Xlsx file with the following parameters:

procedure ExportDBGridEhToXlsx(DBGridEh: TCustomDBGridEh; const FileName: String; Options: TDBGridEhExportAsXlsxOptions; IsSaveAll: Boolean = True);

Where Options is a combination of values

 xlsxColoredEh Export cell and background colors.

 xlsxDataAsDisplayText Unload all cell values as text.

xlsxDataAsEditText Unload all cell values as text. Use text in edit mode. Those. don't use DisplayFormat.

The new version of the library added two procedures for advanced control of data formatting when unloading data to an Xlsx file.

**procedure** ExportDBGridEhToXlsx(DBGridEh: TCustomDBGridEh; **const** FileName: String; ExportOptions: TDBGridEhXlsMemFileExportOptions); **overload**;

**procedure** ExportDBGridEhToXlsx(DBGridEh: TCustomDBGridEh; **const** FileName: String; ExportOptions: TDBGridEhXlsMemFileExportOptions; ExporterClass: TDBGridEhToXlsMemFileExporterClass); **overload**;

 Parameter ExportOptions: TDBGridEhXlsMemFileExportOptions

The new version of the procedure uses the ExportOptions parameter of type TDBGridEhXlsMemFileExportOptions instead of the Options parameter.

The TDBGridEhXlsMemFileExportOptions class contains the following properties:

**property** IsExportSelecting: Boolean

Export only selected area of the grid. If property = False the entire grid will be exported.

**property** ExportColumns: TColumnsEhList

List of columns to export. If you need to export all visible columns of the grid, leave the ExportColumns list empty.

**property** IsExportTitle: Boolean

Whether to export the column headings.

**property** IsExportFooter: Boolean

Whether the grid footer needs to be exported.

**property** IsExportFontFormat: Boolean

Whether it is necessary to export the font format: Name, style, color.

**property** IsExportFillColor: Boolean

Whether to export cell colors.

**property** IsExportCellFormat: Boolean

Whether to export the format of the cells. Vertical and horizontal alignment.

**property** IsExportDisplayFormat: Boolean

Whether to export the format of numbers and dates specified in the Column.DisplayFormat or NumberField/DateField.DisplayFormat property

**property** IsCreateAutoFilter: Boolean

Whether to create an autofilter region.

**property** IsExportFreezeZones: Boolean

Whether to export fixed, non-scrollable zones. Zones are created based on the Grid Header and Frozen columns.

**property** IsFooterSumsAsFormula: Boolean

Whether it is necessary to convert the calculated Sum and Count values into Excel formulas before exporting the footer.

**property** IsExportDataGrouping: Boolean

Whether to export the grouping structure of the grid records.

**property** GridHeaderText: String

General grid header.

**property** GridHeaderFont: TFont;

Grid common header font.

**property** GridHeaderFontStored: Boolean;

Grid common header font is assigned. If the property value is False, then the default font used by Excel will be used. When the GridHeaderFont property is changed, the value of this property is automatically changed to True.

**property** GridFooterText: String;

General grid footer. The text data is added to the Worksheet in the grid data.

**property** GridFooterFont: TFont;

The grid's common footer font.

**property** GridFooterFontStored: Boolean;

The grid's common footer font is assigned.

**property** SheetName: String;

Worksheet page name.

Parameter ExporterClass: TDBGridEhToXlsMemFileExporterClass

This parameter must point to a class inherited from TDBGridEhToXlsMemFileExporter. Calling the ExportDBGridEhToXlsx procedure with the given parameters should be used when it is necessary to change the export algorithm or formatting details when exporting data from DBGridEh to an Xlsx file. Before calling the procedure, write an inheritor from the TDBGridEhToXlsMemFileExporter class and override the necessary virtual functions responsible for the necessary export details. When calling the ExportDBGridEhToXlsx procedure, pass a pointer to the inherited class. The procedure will create an instance of the specified class and will use it to export data.

Here is an example of calling the ExportDBGridEhToXlsx procedure without the ExporterClass parameter:

**procedure** TForm1.actExportToExcelExecute(Sender: TObject);

**var**

 Grid: TDBGridEh;

 Path: String;

 FileName: String;

 ExportOptions: TDBGridEhXlsMemFileExportOptions;

**begin**

 Grid := TDBGridEh(ActiveControl);

 GetDir(0, Path);

 FileName := Path + '\DBGridEhAsXlsx.Xlsx';

 ExportOptions := TDBGridEhXlsMemFileExportOptions.Create;

 ExportOptions.IsExportAll := True;

 ExportOptions.IsExportTitle := True;

 ExportOptions.IsExportFooter := True;

 ExportOptions.IsExportFontFormat := True;

 ExportOptions.IsExportFillColor := True;

 ExportOptions.IsCreateAutoFilter := True;

 ExportOptions.IsExportFreezeZones := True;

 ExportOptions.IsFooterSumsAsFormula := True;

 ExportOptions.IsExportDisplayFormat := True;

 ExportOptions.IsExportDataGrouping := True;

 ExportOptions.SheetName := 'Text';

 ExportOptions.GridHeaderText := 'GridHeaderText';

 ExportOptions.GridHeaderFont := Grid.Font;

 ExportOptions.GridHeaderFont.Size := 24;

 ExportOptions.GridFooterText := 'GridFooterText'

 ExportDBGridEhToXlsx(Grid, FileName, ExportOptions);

 ExportOptions.Free;

 ShellExecute(Handle, nil, PChar(FileName), nil, nil, SW\_SHOWNORMAL);

**end**;

Data export result:



The new version also adds procedures for unloading data into the intermediate TXlsMemFileEh object.

**procedure** ExportDBGridEhToXlsMemFile(DBGridEh: TCustomDBGridEh;

 XlsFile: TXlsMemFileEh; ExportOptions: TDBGridEhXlsMemFileExportOptions);

**procedure** ExportDBGridEhToXlsMemFile(DBGridEh: TCustomDBGridEh; XlsFile: TXlsMemFileEh; ExportOptions: TDBGridEhXlsMemFileExportOptions; ExporterClass: TDBGridEhToXlsMemFileExporterClass);

You can use these functions to unload data first into an object of type TXlsMemFileEh, note to make additional changes in the XlsMemFile object, and after that save the data to a file on disk.

## Exporting data from DBGridEh to TXlsMemFileEh using the TDBGridEhToXlsMemFileExporter class.

By using the TDBGridEhToXlsMemFileExporter class directly, you gain even more flexibility when exporting data to an XlsMemFile and then saving the data to an Xlsx file.

For example, the TDBGridEhToXlsMemFileExporter class has properties such as Worksheet, FromCol, and FromRow. Using the Worksheet property, you can specify which specific workbook to export to. The FromCol and FromRow properties set the starting column and row from which to start generating the export data.

Having written an inheritor from TDBGridEhToXlsMemFileExporter, you can overload the virtual methods of the class to change the format of data export.

Below is the implementation of the ExportDBGridEhToXlsMemFile procedure, which you can understand how to use the TDBGridEhToXlsMemFileExporter class.

**procedure** ExportDBGridEhToXlsMemFile(DBGridEh: TCustomDBGridEh; XlsFile: TXlsMemFileEh;

 ExportOptions: TDBGridEhXlsMemFileExportOptions; ExporterClass: TDBGridEhToXlsMemFileExporterClass);

**var**

 Exporter: TDBGridEhToXlsMemFileExporter;

**begin**

 **if** ExporterClass <> nil

 **then** Exporter := ExporterClass.Create

 **else** Exporter := TDBGridEhToXlsMemFileExporter.Create;

 Exporter.XlsFile := XlsFile;

 Exporter.Grid := DBGridEh;

 **if** ExportOptions <> nil **then**

 Exporter.ExportOptions := ExportOptions;

 Exporter.ExportGrid;

 Exporter.Free;

**end**;

Here in the code:

If ExporterClass is not specified, then the default TDBGridEhToXlsMemFileExporter class is used.

The XlsFile, DBGridEh and ExportOptions object is assigned.

Exporter.ExportGrid method exports data.

Once exported, the Exporter object is no longer needed and is removed. The generated data remains in the XlsFile class.

The TDBGridEhToXlsMemFileExporter.Worksheet property is not assigned to the code. Therefore, the Exporter renders data on the first workbook from the Workbook.Worksheets collection.

# ***Extended version of procedures for unloading data from DBGridEh to text file.***

In version 9.5, there was an unloading of DBGridEh into text views, a set of classes TDBGridEhExportAsText, TDBGridEhExportAsUnicodeText, TDBGridEhExportAsCSV with the SaveDBGridEhToExportFile procedure was used.

EhLib 10.0 proposes to use one TDBGridEhTextExportOptions class with the SaveDBGridEhToTextFile procedure.

The simplest call to the SaveDBGridEhToTextFile procedure looks like this:

**procedure** TfrImportExport.btnExportAsTxtClick(Sender: TObject);

**var**

 Path: string;

 FileName: string;

 ExportOptions: TDBGridEhTextExportOptions;

**begin**

 ExportOptions := TDBGridEhTextExportOptions.Create;

 ExportOptions.IsExportTitle := True;

 ExportOptions.IsExportFooter := False;

 ExportOptions.Encoding := TEncoding.UTF8;

 ExportOptions.WriteBOM := True;

 ExportOptions.QuoteChar := '"';

 ExportOptions.CellDelimiter := #09;

 ExportOptions.TrailingLineDelimiter := True;

 ExportOptions.UseEditFormat := False;

 GetDir(0,Path);

 FileName := Path + '\DBGridEh1Export.Txt';

 SaveDBGridEhToTextFile(DBGridEh1, FileName, ExportOptions);

 ExportOptions.Free;

 ShellExecute(Handle, nil, PChar(FileName), nil, nil, SW\_SHOWNORMAL);

**end**;

In this example, before calling the SaveDBGridEhToTextFile procedure, the TDBGridEhTextExportOptions class is created in which the settings for exporting grid data are written.

The TDBGridEhTextExportOptions class has extensive properties with which to control the size of the data export format.

You can also use the following procedures and functions to export data.

**procedure** SaveDBGridEhToTextFile(DBGridEh: TCustomDBGridEh; **const** FileName: String; ExportOptions: TDBGridEhTextExportOptions);

Saving grid data to a file as text.

**procedure** WriteDBGridEhToTextStream(DBGridEh: TCustomDBGridEh; Stream: TStream; ExportOptions: TDBGridEhTextExportOptions);

Exporting grid data to a stream in text format.

**function** WriteDBGridEhToString(DBGridEh: TCustomDBGridEh; ExportOptions: TDBGridEhStringExportOptions): String;

Exporting grid data to a string.

TDBGridEhStringExportOptions class contains the following properties

**property** ExportSelecting: Boolean;

Specifies to unload only the selected grid area. If property = False, then the entire grid will be unloaded regardless of the presence of the selected MultiSelect area.

**property** CellDelimiter: String;

A string separator of cells. Default #9 (Tab code).

**property** LineDelimiter: String;

Line separator string. The default is sLineBreak.

**property** TrailingLineDelimiter: Boolean;

The property specifies whether to unload the line separator after the last unloaded line.

**property** QuoteChar: Char;

Use QuoteChar to get or set the quote character that is used to enclose individual cell values if the value contains a substring that matches the value of the CellDelimiter, LineDelimiter, or QuoteChar property.

**property** IsExportTitle: Boolean;

The property specifies whether to write the column headings in the first row of the export.

**property** IsExportFooter: Boolean;

The property specifies whether to unload grid footers after unloading the split data.

**property** UseEditFormat: Boolean;

The property specifies that when receiving cell data as a string for numeric data and DateTime data, the string format used when editing the cell must be used. If property = False, then Column.DisplayFormat property will be used for formatting.

**property** FormatSettings: TFormatSettings;

The property specifies the format for numeric and DateTime values to be used when converting numbers and dates to text.

**property** UseFormatSettings: Boolean;

Set the property to True to indicate that the FormatSettings property should be used when converting numeric data.

**property** ExportColumns: TColumnsEhList;

List of columns to be exported. If the property is not filled, then all visible rows of the grid will be exported.

The TDBGridEhTextExportOptions class inherits from the TDBGridEhStringExportOptions class and has the following additional properties.

**property** Encoding: TEncoding;

The property specifies the character encoding that will be used when writing the exported data string to a stream or file.

**property** WriteBOM: Boolean;

Set the property to True to indicate to the algorithm that when writing data to a stream, a Byte Order Mark must be written to the beginning of the stream, which contains the encoding code in which the text data was written. For a description of the Byte Order Mark, see here: <https://en.wikipedia.org/wiki/Byte_order_mark>

# ***Search history in DBGridEh.SeachPanel***

Added MRUList property to DBGridEh.SeachPanel. This property allows you to customize the history of “Most Recently Used values” of search values in the text editor of the SeachPanel section.



To activate the list, set the DBGridEh.SeachPanel.MRUList.Active property to True.

For a description of the other subproperties of the DBGridEh.SeachPanel.MRUList property, see “Working with lists MRUList (last entry list)”.

By default, new text editor values are added to the list when the user presses Enter or the text editor loses focus.

# ***High DPI monitors support***

The new version has improved support for displaying data on high-resolution monitors (High DPI displays). The data is scaled using the TControl.ScaleFactor property added in VCL XE10.3.



# ***Changes in the interface of the library modules.***

---

IXlsFileCellsRangeEh.InsideBorder property removed.

Use the IXlsFileCellsRangeEh.Border.InsideHorizontal and IXlsFileCellsRangeEh.Border.InsideVertical properties instead.

---

When calling the MergeCell procedure

procedure TXlsWorksheetEh.MergeCell(Col, Row, ColCount, RowCount: Integer);

the ColCount and RowCount parameters must contain the number of columns and rows given the underlying column.

Those. to merge two cells horizontally, call the method with parameters

Sheet.MergeCell(Col, Row, 2, 1);

In version 9.5, similar code looked like:

Sheet.MergeCell(Col, Row, 1, 0);

---

The TFooterValues type has been moved to the DBGridEh module

---

The TDBGridMultiTitleExportNodeEh type has been moved to the DBGridEh module

---

The TDBGridMultiTitleExportNodeMatrixEh type has been moved to the DBGridEh module

---

The DBGridEhXlsMemFileExporters module has been added to the library. The module contains the TDBGridEhToXlsMemFileExporter class, which is designed to export the TDBGridEh component to an object of the TXlsMemFileEh type and then save the data to the Xlsx file.

---

The XlsFileReadersEh module has been added to the library. The module contains the TXlsFileReaderEh class, which is designed to load data from Xlsx files into an object of the TXlsMemFileEh type. The TXlsFileReaderEh class is used in the TXlsMemFileEh.LoadFromFile method.

---

In DBGridEh, the smstClassicEh type marker is now displayed as smstFrameEh, because picture for smstClassicEh does not scale well for HighDPI monitors.

# ***Working with the TXlsMemFileEh class***

The TXlsMemFileEh class is designed to work with files in the Xlsx format.

TXlsMemFileEh allows loading data from Xlsx files, changing content and saving changed data to Xlsx file.

The TXlsMemFileEh class does not use external libraries or applications and works with Xlsx files directly through the internal library code.

An xlsx file is essentially a compressed Zip archive. The archive contains a set of files in Xml format.

Using standard VCL procedures, the TXlsMemFileEh class has the ability to unpack Xlsx files and parse the Xml files of the archive.

As a result of parsing TXlsMemFileEh creates the following objects and properties in itself.

## Property Workbook: TXlsWorkbookEh

Workbook. TXlsMemFileEh contains one copy of a workbook of type TXlsWorkbookEh. The workbook contains a set of tables (TXlsWorksheetEh) and TXlsFileStylesEh styles.

Styles describe the format of the data cells in the Worksheet. Each cell must reference one of the styles from the Styles list. When filling in data in the program code, you do not need to use the TXlsWorkbookEh.Styles property. The inner code of the TXlsMemFileEh class automatically creates styles and style references when the format changes in table cells.

## Class properties and methods TXlsWorkbookEh

The following TXlsWorkbookEh properties and methods are used to work with work pages (Worksheet):

 **function** AddWorksheet(WorksheetName: string): TXlsWorksheetEh;

Add a new table (Worksheet) with the specified name.

 **function** FindWorksheet(WorksheetName: string): TXlsWorksheetEh;

Find a table (Worksheet) by name.

 **procedure** MoveWorksheet(FromIndex, ToIndex: Integer);

Move the Worksheet from the FromIndex position to the ToIndex position.

 **procedure** RemoveWorksheet(WorksheetEh: TXlsWorksheetEh);

Delete Worksheet.

 **property** Worksheets[WorksheetId: Variant]: TXlsWorksheetEh;

Get a ref to a table by name or by index. To get a value by name, pass the table name as a string to the WorksheetId parameter.

 **property** WorksheetCount: Integer;

Get the number of tables in the Worksheets collection.

## TXlsWorksheetEh Class

TXlsWorksheetEh class contains tabular data for one table.

Tabular data includes:

Column definition (Properties Columns, DefaultColWidth)

Defining strings (Properties Rows, DefaultRowHeight)

Values in Table Cells (Cells Property)

Format for each cell (Color, Font, Line thickness, etc.) (Via IXlsFileCellsRangeEh interface)

Define row grouping (Set via the properties Rows, OutlineRowsSummaryBelow)

Column grouping definition (Set via properties Columns, OutlineColsSummaryRight)

Print settings (Subproperties properties PrintParams)

Setting non-scrollable areas (Properties FrozenColCount, FrozenRowCount)

TXlsWorksheetEh class has the following properties:

**property** Name: String;

Worksheet title. In Excel, the title appears at the bottom of the bookmark list.

**property** TabColor: TColor;

Table bookmark color.

**property** AutoFilterRange: TXlsFileWorksheetCellsRectEh;

Auto filter area

**property** CellDataExists[Col, Row: Integer]: Boolean;

Determine if the TXlsFileCellEh object was created at the specified coordinates.

**property** Cells[Col, Row: Integer]: TXlsFileCellEh;

Access to table cells with coordinates Col, Row. Coordinates start at zero.

**property** Columns: TXlsFileColumnsEh;

Page column options.

**property** DefaultColWidth: Double;

The default column width.

**property** DefaultRowHeight: Double;

Default line height.

**property** DefaultRowHeight: Double;

The size of the area that contains nonblank cells

**property** FrozenColCount: Integer;

The number of columns frozen.

**property** FrozenRowCount: Integer;

The number of frozen lines.

**property** OutlineRowsSummaryBelow: Boolean;

The group headers for the columns are below the group. The property is used when grouping strings.

**property** OutlineColsSummaryRight: Boolean

The row group headers are located to the right of the group. The property is used when grouping columns.

**property** PrintParams: TXlsFileWorksheetPrintParamsEh;

Print Options: Page Orientation, Header, etc.

**property** Rows: TXlsFileRowsEh;

Page line options.

**property** ZoomScale: Integer;

Scaling. Set as a percentage. The default is 100.

## Definition of table columns.

Use the Columns collection to define properties for table columns.

The collection does not require adding columns to define the list of columns. Columns in a collection are created automatically when accessing collection items.

For example the code:

Sheet.Columns[5].Width := 20;

Creates a list of five columns in the collection (if they are not already in the collection) and assigns the heel of the column a width of 20 Width Units. Column numbering in the collection starts from zero.

The size of the unit column width in Excel is set according to the following rule:

Column width measured as the number of characters of the maximum digit width of the numbers 0, 1, 2, …, 9 as rendered in the normal style's font. There are 4 pixels of margin padding (two on each side), plus 1 pixel padding for the gridlines.

Use TXlsFileColumnsEh.ScreenToXlsWidth function to convert screen column width to width in Excel units.

Sheet.Columns[1].Width := Sheet.Columns.ScreenToXlsWidth(DBGridEh1.Columns[1].Width);

The Sheet.Columns collection contains objects of type TXlsFileColumnEh.

TXlsFileColumnEh class contains the following properties:

**property** OutlineLevel: Integer;

Grouping level. The property is used when grouping columns in a table. If column does not participate in the grouping, then the value contains 0.

**property** OutlineNodeCollapsed;

The group header is collapsed. The property is used when grouping columns in a table. If the column is a group header and the group is collapsed, then the property contains True. Columns that are in a collapsed group also need to set the Visible property to False.

**property** Width: Double;

Column width in Excel units of width.

**property** Visible: Boolean;

The column is visible.

## Defining strings (Properties Rows, DefaultRowHeight)

Use the Rows collection to define the properties of the table rows.

The collection does not require adding strings to define a list of strings. The rows in the collection are created automatically when accessing the items in the collection.

Working with table rows is similar to working with columns.

## Values in table cells (Property Cells)

Use the Sheet.Cells two-dimensional array property to set cell values.

The array does not require adding cells to expand the size of the array. The array elements are created automatically when accessed.

The cell values are set by the Variant property Value. The property supports numeric, string, boolean data types, and the DateTime type for specifying the date and time.

Below is an example of setting table values

Sheet.Cells[0, i + 4].Value := DBGridEh1.Columns[0].Field.Value;

Sheet.Cells[1, i + 4].Value := DBGridEh1.Columns[1].Field.Value;

Sheet.Cells[2, i + 4].Value := DBGridEh1.Columns[2].Field.Value;

In addition to values, a cell can also contain a formula. Use the Formula property to define the formula.

Below is an example of setting a formula for one cell of a table

Sheet.Cells[1, i+4].Formula := 'SUM(E5:' + 'E' + IntToStr(i+4) + ')';

TXlsMemFileEh class does not perform formula cell calculations. Setting the Formula property does not change the Value property. However, if you create a file that you plan to open in MS Excel, then when you open the file in MS Excel, the user will see the calculated data, because MS Excel recalculates the value of formulas when opening a file.

## Formatting cells in a table.

Cell formatting refers to setting the Color, Font, Separating Lines, Alignment, Indent, Digital format for one or more data cells.

The cells are formatted using the GetCellsRange method and the IXlsFileCellsRangeEh interface.

Using the Sheet.GetCellsRange method, you need to get a reference to the specified array of cells in the table (GetCellsRange returns an array of cells as a reference to the IXlsFileCellsRangeEh interface).

Then, use the methods of the IXlsFileCellsRangeEh interface to perform the cell format changes and call the IXlsFileCellsRangeEh.ApplyChages method to apply the changes to the table.

The following code gets a reference to a range of four cells (LeftCol = 0, TopRow = 0, ColCount = 1, RowCount = 1) and sets the font size for the selected range to 24.

**var**

 cr: IXlsFileCellsRangeEh;

**begin**

 cr := XlsFile.Workbook.Worksheets[0].GetCellsRange(0, 0, 1, 1);

 cr.Font.Size := 24;

 cr.ApplyChages;

**end**;

Use the following IXlsFileCellsRangeEh interface properties to set other format properties.

**property** Font: TXlsFileCellsRangeFontEh;

Sets the font properties for the selected range.

**property** Fill: TXlsFileCellsRangeFillEh;

Sets the cell shading properties for the selected range.

**property** Border: TXlsFileCellsRangeLinesEh;

Sets the properties of the colors and the width of the dividing lines of the cells for the selected range.

**property** HorzAlign: TXlsFileCellHorzAlign;

Sets the horizontal text in cells for the selected range.

**property** VertAlign: TXlsFileCellVertAlign;

Specifies the vertical alignment of text in cells for the selected range.

**property** WrapText: Boolean;

Specifies whether the text should be wrapped but a new line if the text does not fit into the width of the cell.

**property** Rotation: Integer; //Degrees

Specifies the rotation of the text.

**property** Indent: Integer;

Sets the indentation of the text to the left inside the cell.

**property** CharsFlowDirection: TXlsFileCharsFlowDirectionEh;

Specifies how the letters of the text are positioned relative to each other.

**property** NumberFormat: String **read** GetNumberFormat **write** SetNumberFormat;

Specifies the number format for cells that contain numbers or date / time. The format is set according to the rules of MS Excel.
For a description of the number format, see the MS Excel documentation.
<https://support.microsoft.com/en-us/office/number-format-codes-5026bbd6-04bc-48cd-bf33-80f18b4eae68>

## Setting grouping in rows and columns of a table.

To set up grouping rows in a table, you must correctly fill in the OutlineLevel and OutlineNodeCollapsed properties of the TXlsFileRowEh class in the Worksheet.Rows collection or Worksheet.Columns if you are grouping columns.

OutlineLevel sets the grouping level of the entry.

In the screenshot below, the values that must be assigned to the OutlineLevel property for each table record are indicated in red.



Entries **1** and **9** have grouping level zero.

Record number **1** is a table header and does not participate in the grouping.

Record number 9 is a level 1 group heading. The level heading must have a value in OutlineLevel one less than the level of the group it describes (Group Heading 1 - 1 = 0)

Records 4 and 8 are headers for the level 2 group. Accordingly, OutlineLevel contains the value 1.

For records 4, 8 and 9, you can also specify that the group is collapsed. To do this, set the OutlineNodeCollapsed property to True. In this case, also, for records that are included in the collapsed group, you must set the Visible property to False.

If the group header records are above the group, then the TXlsWorksheetEh.OutlineRowsSummaryBelow property must be set to False.

The OutlineLevel and OutlineNodeCollapsed properties for the TXlsFileColumnEh classes are filled in the same way when grouped by columns.

## Combining (Merging) cells.

Use the TXlsWorksheetEh.MergeCell and TXlsWorksheetEh.UnmergerCell methods to merge cells and remove the merge.

The MergeCell procedure has the following calling interface:

 procedure MergeCell(Col, Row, ColCount, RowCount: Integer);

The Col and Row parameters point to the underlying column and row. Values start at 0.

The ColCount and RowCount parameters must contain the number of columns and rows to combine based on the underlying column.
Those. to merge two cells horizontally, call a method with parameters:

Sheet.MergeCell(Col, Row, 2, 1);

The UnmergerCell procedure has the following call parameters:

procedure UnmergerCell(Col, Row: Integer);