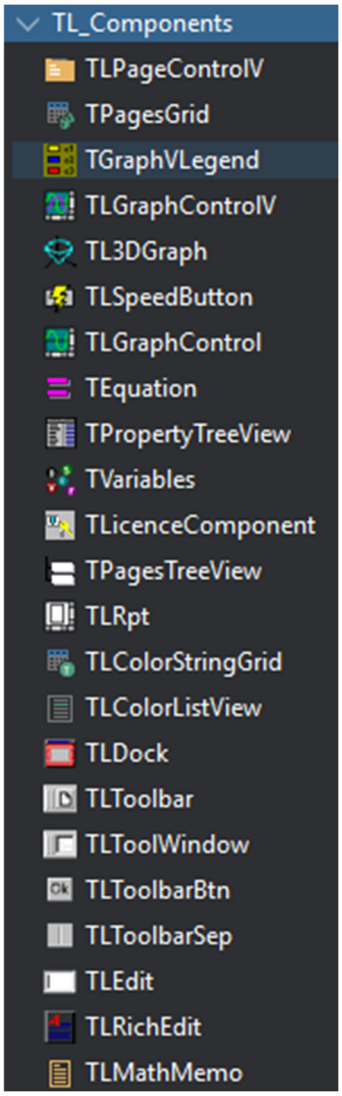
TLComponents

for Delphi



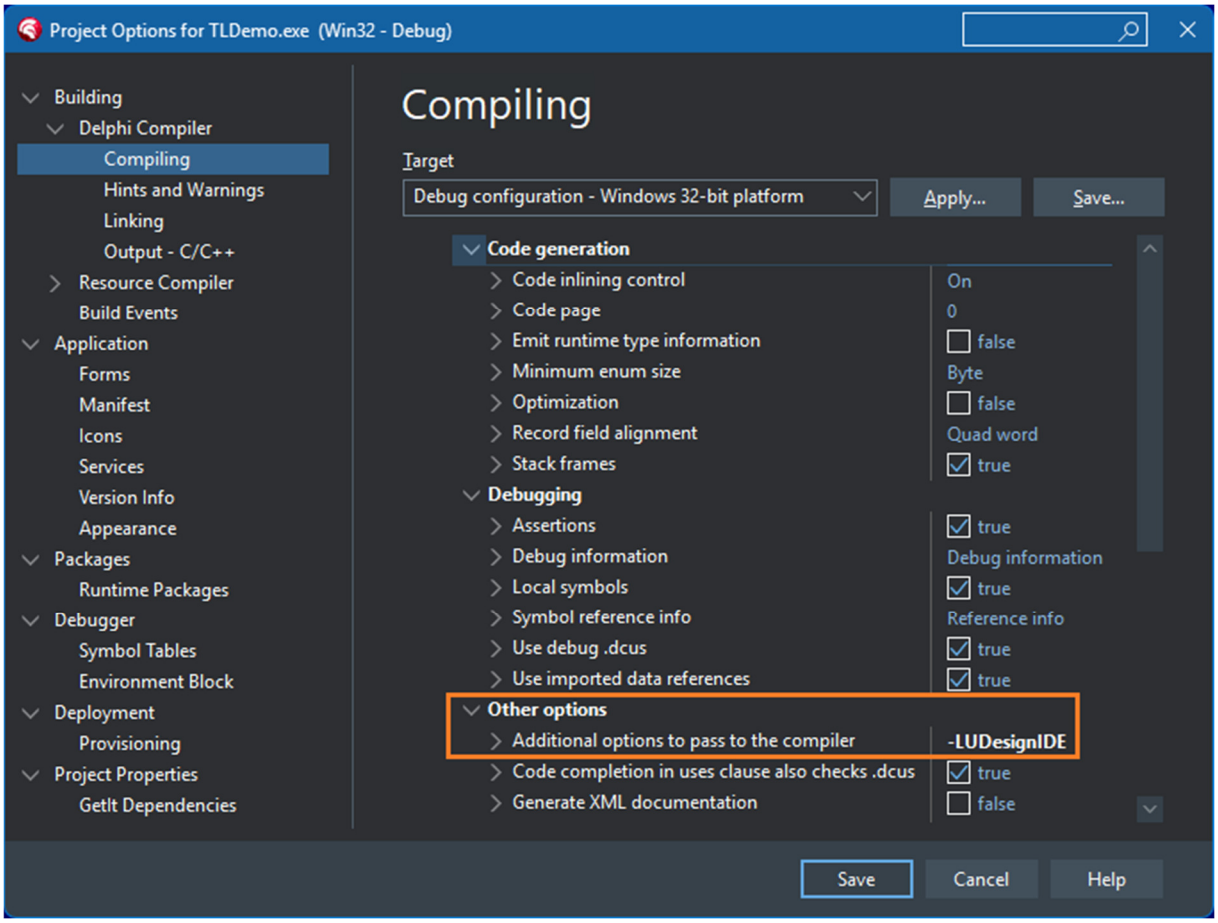
Contents

1. Package TLComponents

The package TLComponents for Delphi contains these components:

TLicenceComponent, TEquation, TLMathMemo, TVariables, TPropertyTreeView, TLRichEdit, TLGraphControl, TLGraphControlV, TGraphVLegend, TL3DGraph, TLPageControlV, TLColorListView, TLColorStringGrid, TLRpt, TPagesGrid, TPagesTreeView, TLSpeedButton, TLDock, TLToolBar, TLToolWindow, TLToolbarBtn, TLToolbarSep, TLEdit.

It is necessary to set the following in the project utilizing TLComponents:



It is necessary to insert the component TLicenceComponent into the main programme window for the full functionality of some components. Subsequently, it is necessary to set the property **LicenceComponent** for these components.

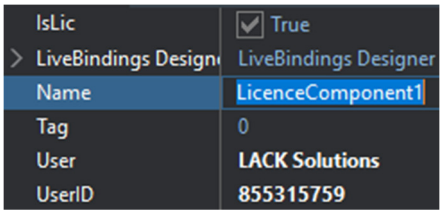
The components are demonstrated in the **TLDemo** application.

The section interface units are in the **Interfaces** directory.

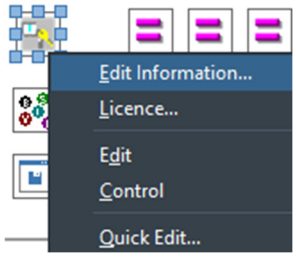
2. Components

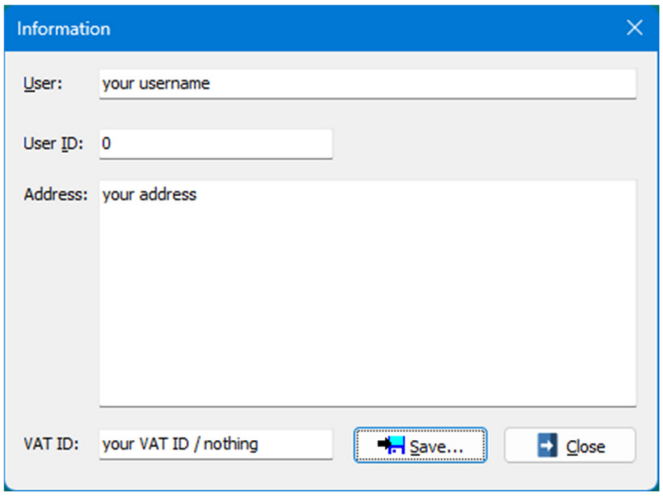
2.1 TLicenceComponent = class(TInfoComponent)

The TLicenceComponent component properties:



For the activation of the licence, it is necessary to generate the file *.inf using the window "Information":





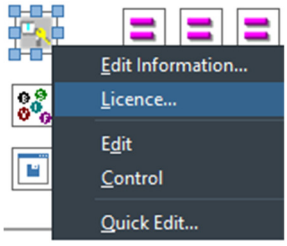
It is necessary to select the item "User" and to fill in the item "Address" and the item "VAT ID". If you do not have VAT ID, then leave the item "VAT ID" empty.

It is necessary to send the generated file *.inf to the address: lacktomas@gmail.com.

Subsequently, an invoice for the licence fee will be sent to you. The package price TLComponents is 170 euros without VAT for the computer on which the *.inf file was generated.

After paying the invoice, a *.lic file will be sent to you.

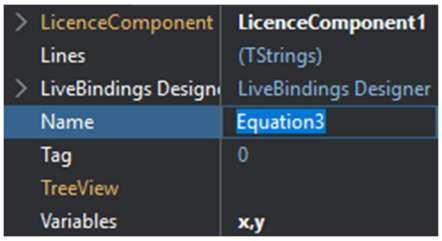
You will load the *.lic file using the "Licence" window:



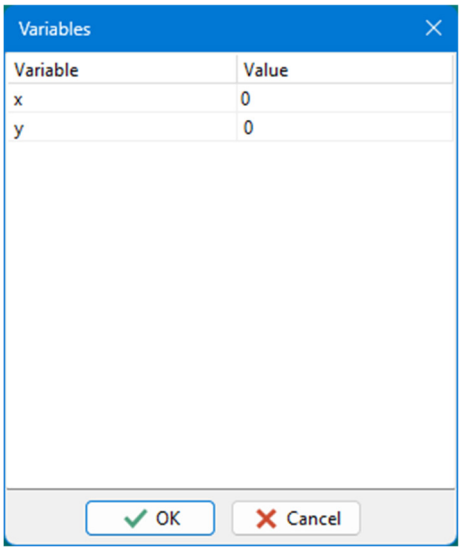
2.2 TEquation = class(TInfoComponent)

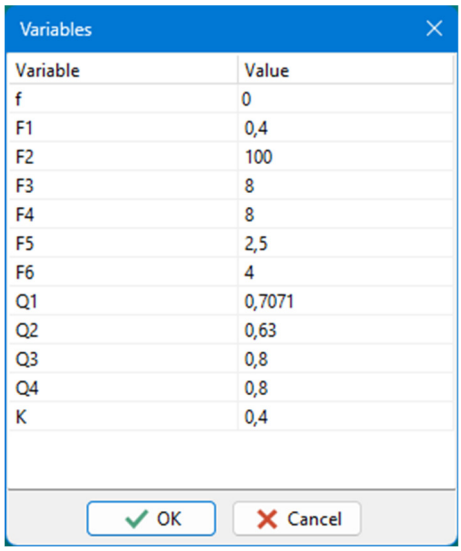
The component TEquation makes it possible to gain the numerical value of mathematical equation given by the text

The TEquation component properties:



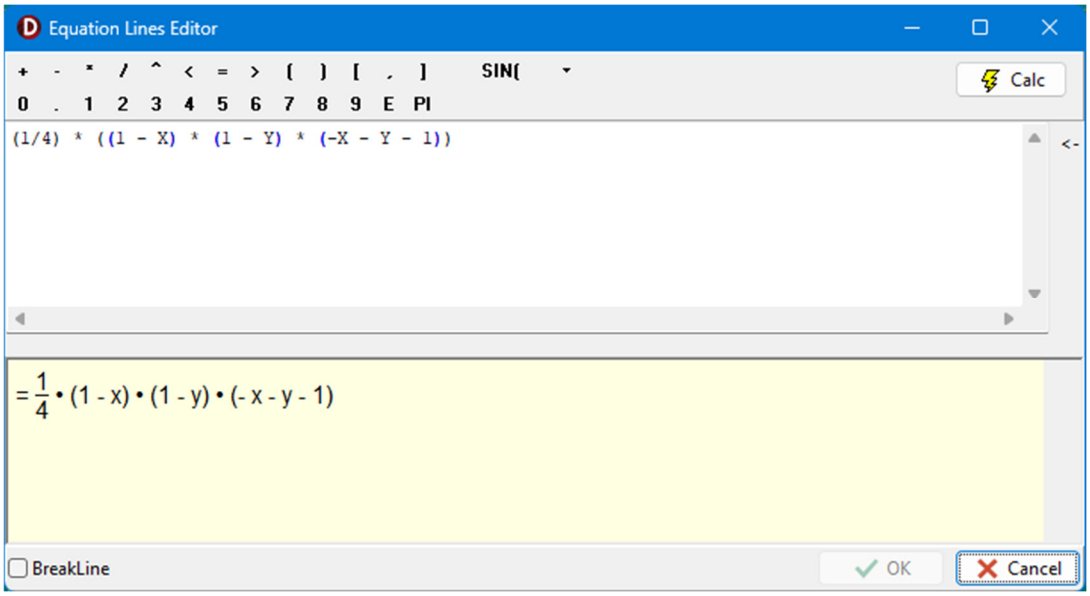
Equations variables or parameters are defined in the "**Variables**" property:





Firstly, it is best to the variables and then the other equation parameters.

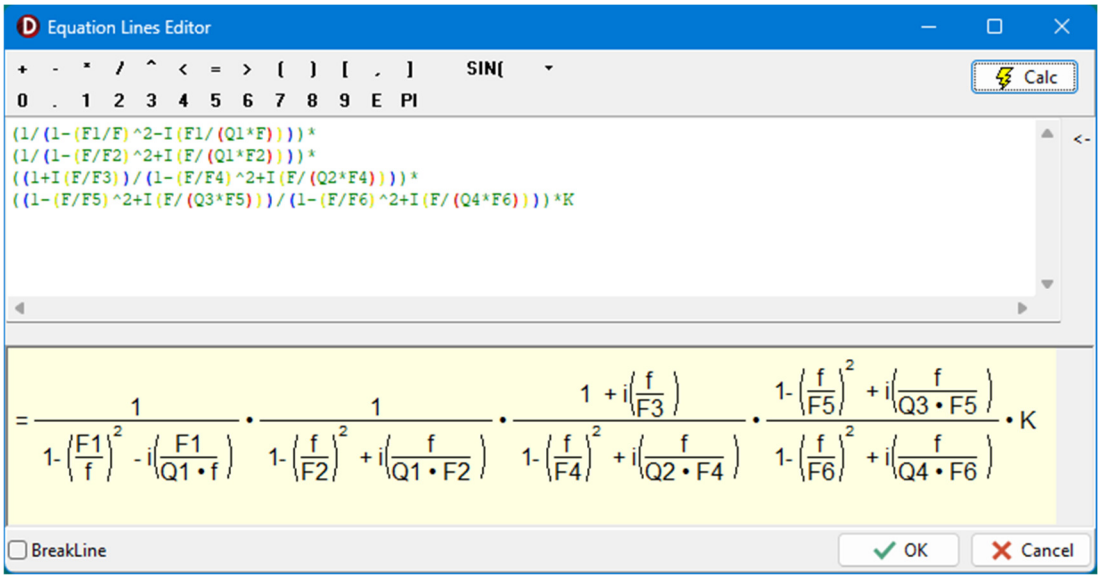
The mathematical equation is defined in the "**Lines**" property:



To gain the numerical value, then it is possible to use the function:

```
z := Equation.GetValue([x, y]).Real;
```

For example, for the complex equation in the "**Lines**" property:

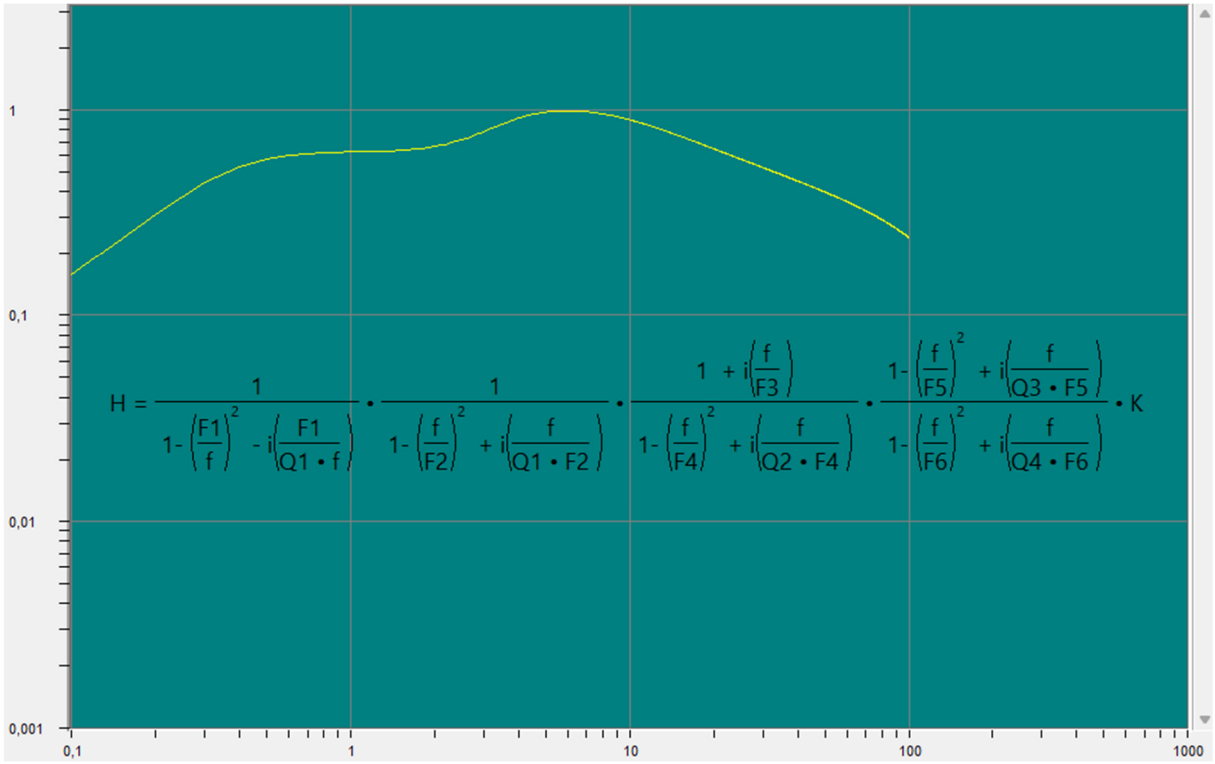


To gain the complex numerical value, then it is possible to use the function:

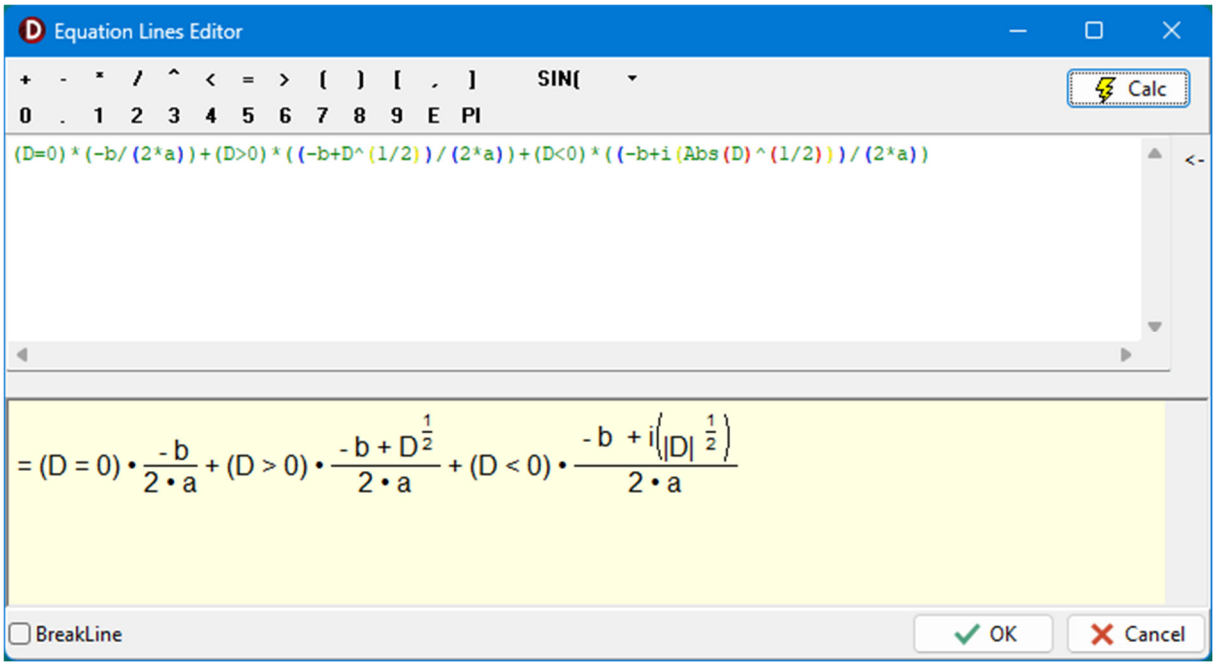
```
C := Equation.GetValue([f{, 0.4, 100.0, 8.0, 8.0, 2.5, 4.0, 0.7071, 0.63, 0.8, 0.8, 0.4}]);
```

If it is needed to change only the variable "f", only the variable "f" will be used into the GetValue function and other value will be taken from the "**Variables**" property.

In the following picture, a display of the absolute value of the H complex function is depicted.



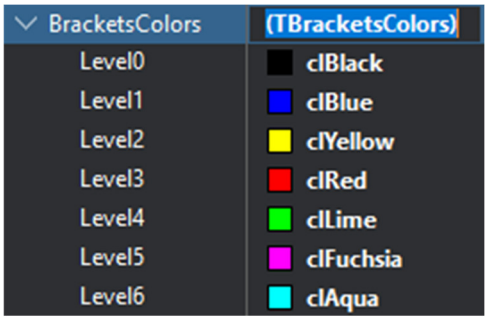
It is possible to use also logical conditions in the equation:



The component TEquation has one event, namely **`OnChange`**: `TNotifyEvent`. This event is executed when an equation text changes.

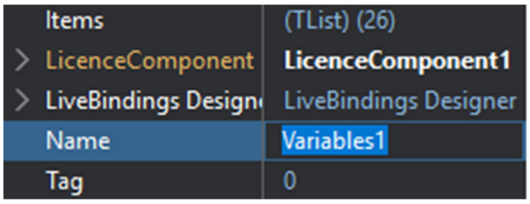
2.3 TLMathMemo = class(TMemo)

The TLMathMemo component is the child of the TMemo component. The TLMathMemo component depicts the brackets in a different color according to the level. The brackets color is determined by the **BracketsColors** property.

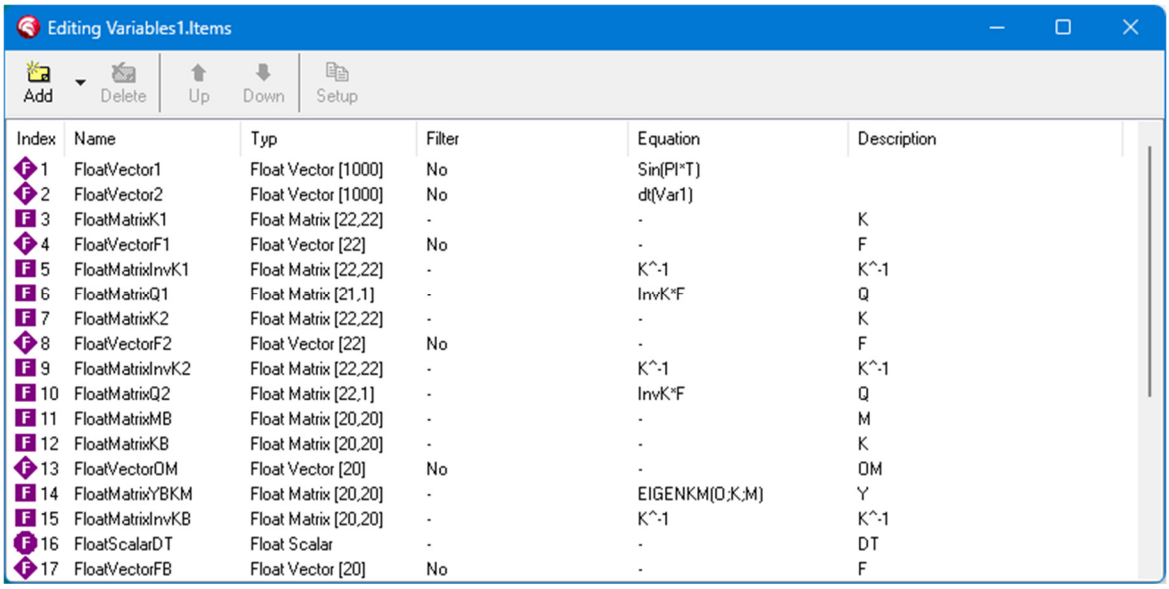


2.4 TVariables = class(TContainer)

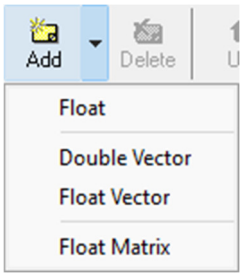
The TVariables component properties:



It is possible to add/edit the individual variables in the "**Items**" property.

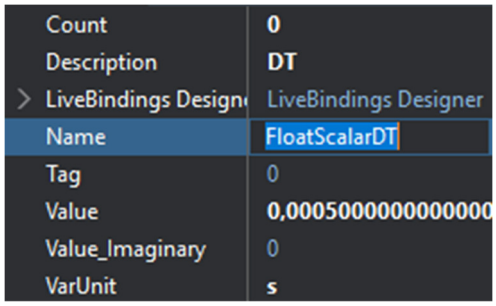


It is possible to add these components:



2.4.1 TFloatScalar = class(TDVariable)

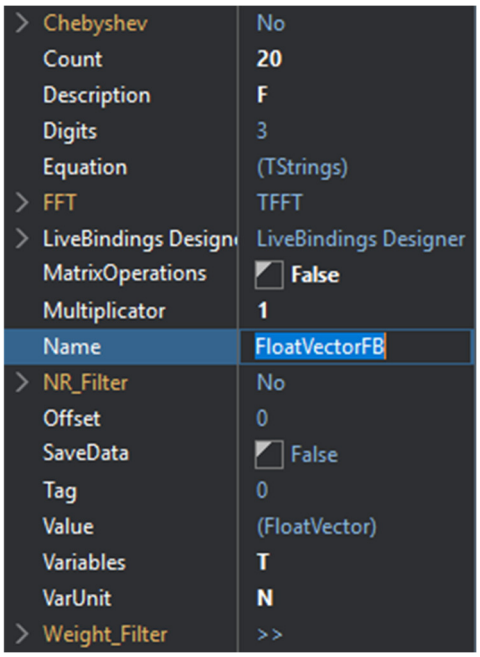
The TFloatScalar component properties:



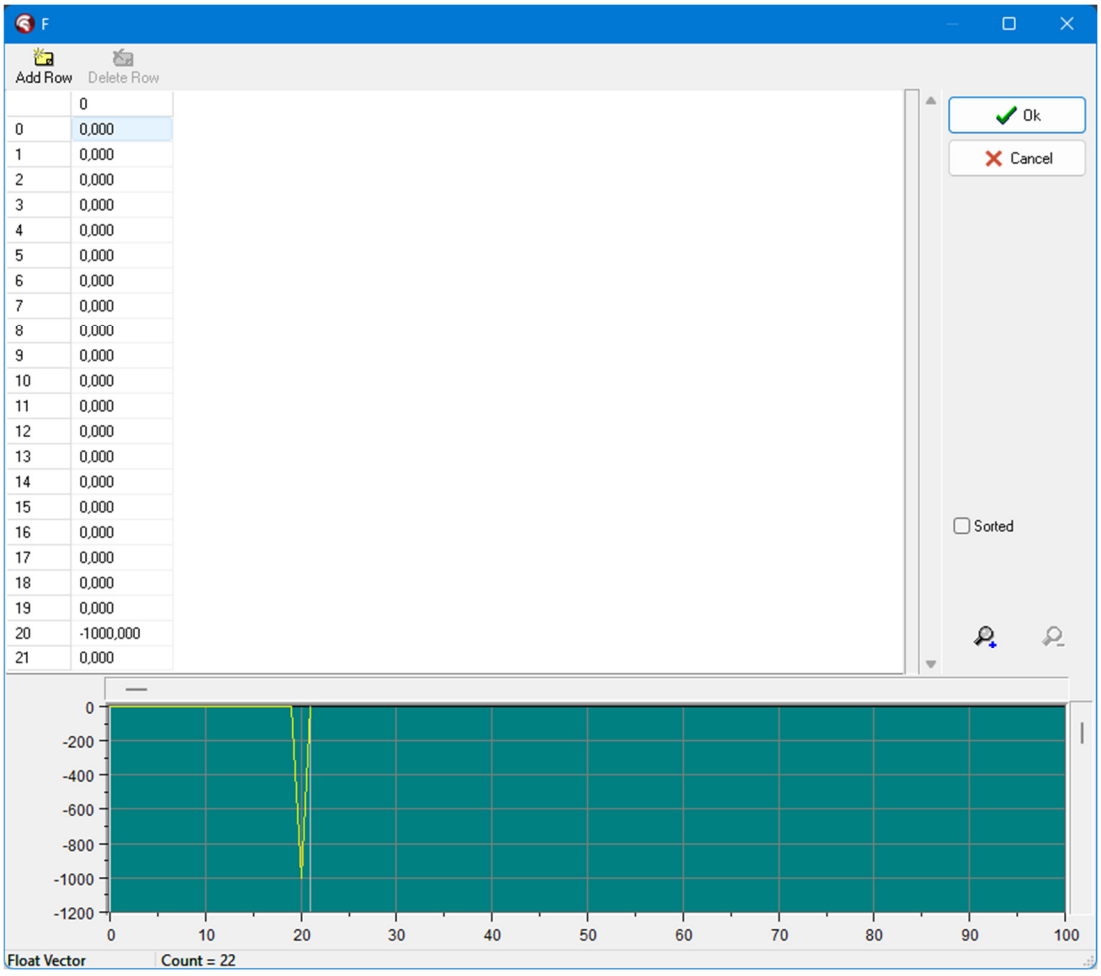
The value of the variable may also be a complex number.

2.4.2 TFloatVector = class(TCustomFloatVector)

The TFloatVector component properties:



It is possible to modify the vector content and to approach its content in three ways. The first option is to edit the content using the editing window:

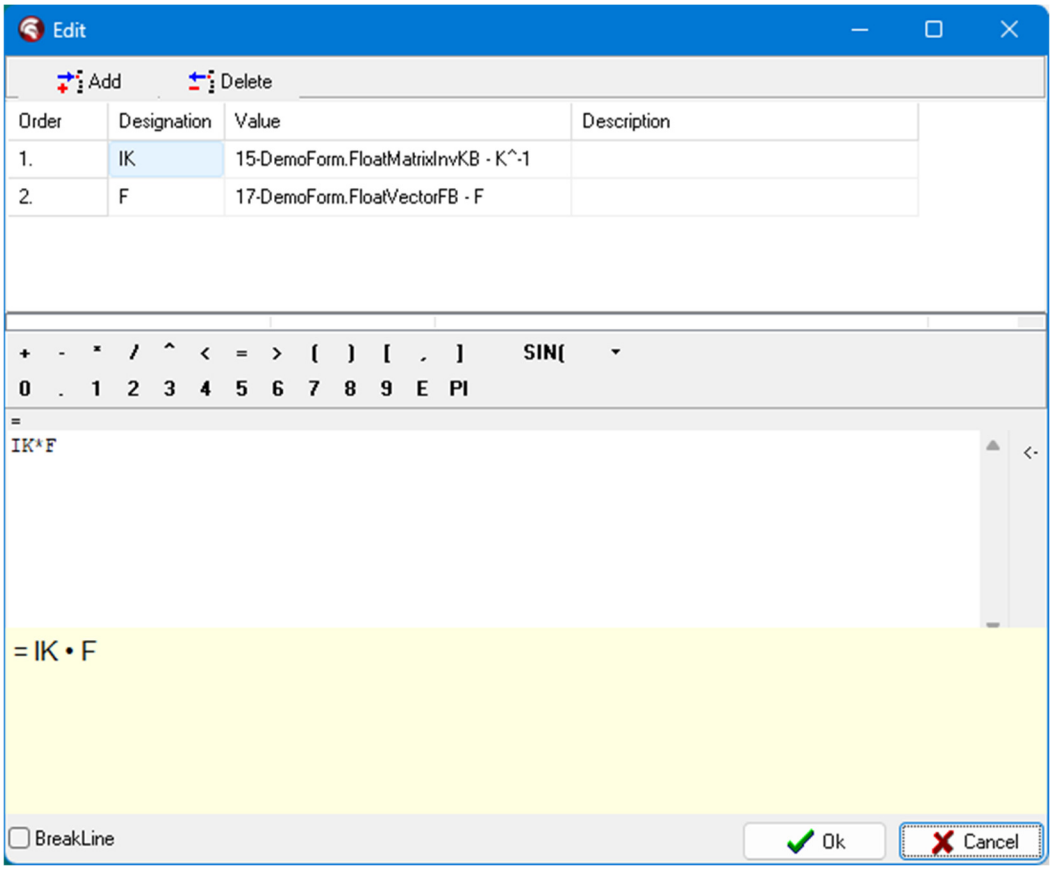


It is necessary to insert the value “True” into the property “**SaveData**” to edit and save the content into the *.dfm file.

The second option is to insert the content / to modify the content and to approach the content in the source code programme using these procedures and properties:

```
    procedure SetSubVector(I: Integer; SubVector: TSingle1D);
    procedure GetComplexSubVector(I: Integer; var SubVector: TComplex1D);
    procedure SetComplexSubVector(I: Integer; SubVector: TComplex1D);
    procedure AddSubVector(I: Integer; SubVector: TSingle1D);
    procedure AddComplexSubVector(I: Integer; SubVector: TComplex1D);
    procedure CopyDataFrom(Source: TCustomFloatVector);
    procedure SetFloatValueAt(Row, Col: LongWord; Value: Extended);
    procedure Sort;
    procedure Clear;
....procedure Load;
    property Data[I: LongWord]: Single read GetData write SetData; default;
    property DataIm[I: LongWord]: Single read GetDataIm write SetDataIm;
    property SortIx[I: Integer]: Integer read GetSortIx;
```

The third option is to calculate the content using the “**Equation**” property:



It is necessary to insert the value "True" into the property "**MatrixOperations**" to enable matrix operations. Then, it is necessary to call the procedure **`CalcEquation`** in the source code.

When using the **Sort** procedure, the content of the vector is not sorted, yet **SortIx** is sorted.

Then, it is possible to gain:

- Minimal value as `Data[SortIx[0]]`
- Maximum value as `Data[SortIx[Count-1]]`
- Median as `Data[SortIx[Count div 2 – 1]]`

The TFloatVector component has these Events:

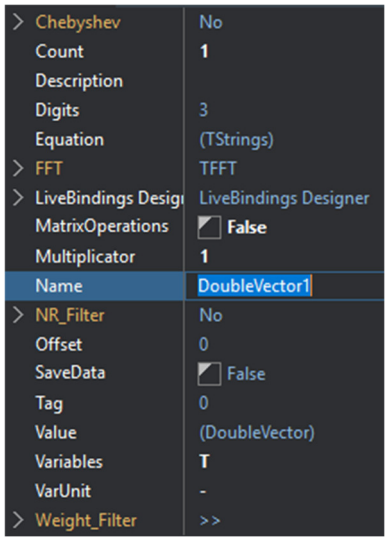
`property OnSetVarValues: TSetVarValues = = procedure (Sender: TObject; I, J: Integer; var VarValues: TExtended1D) of Object;` - is executed when changing VarValues[0] = T (time).

`property OnGetValue: TCustomFloatVectorGetValue = procedure (Sender: TObject; I: Integer; var V: TComplex) of Object;` - is executed from the procedure `Load`.

`property OnAfterCalc: TNotifyEvent;` - is executed from the procedure `EquationCalc`.

2.4.3 TDoubleVector = class(TCustomFloatVector)

The TDoubleVector component properties:



The object TDoubleVector is the same as the object TFloatVector with a difference in data type "Double".

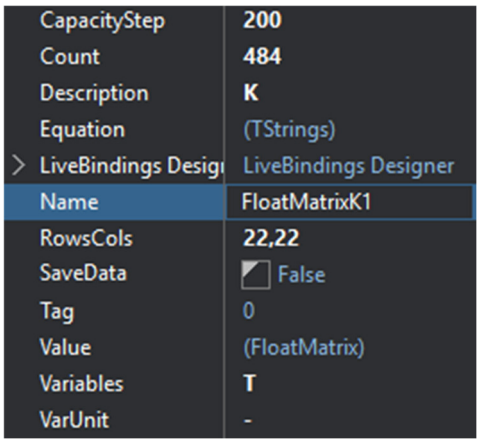
It is possible to insert the content / to modify the content and to approach the content in the source code programme for the TDoubleVector using these procedures and properties:

```
procedure SetSubVector(I: Integer; SubVector: TDouble1D);

procedure GetComplexSubVector(I: Integer; var SubVector: TComplex1D);

procedure SetComplexSubVector(I: Integer; SubVector: TComplex1D);

procedure AddSubVector(I: Integer; SubVector: TDouble1D);

procedure AddComplexSubVector(I: Integer; SubVector: TComplex1D);

procedure CopyDataFrom(Source: TCustomFloatVector);
```

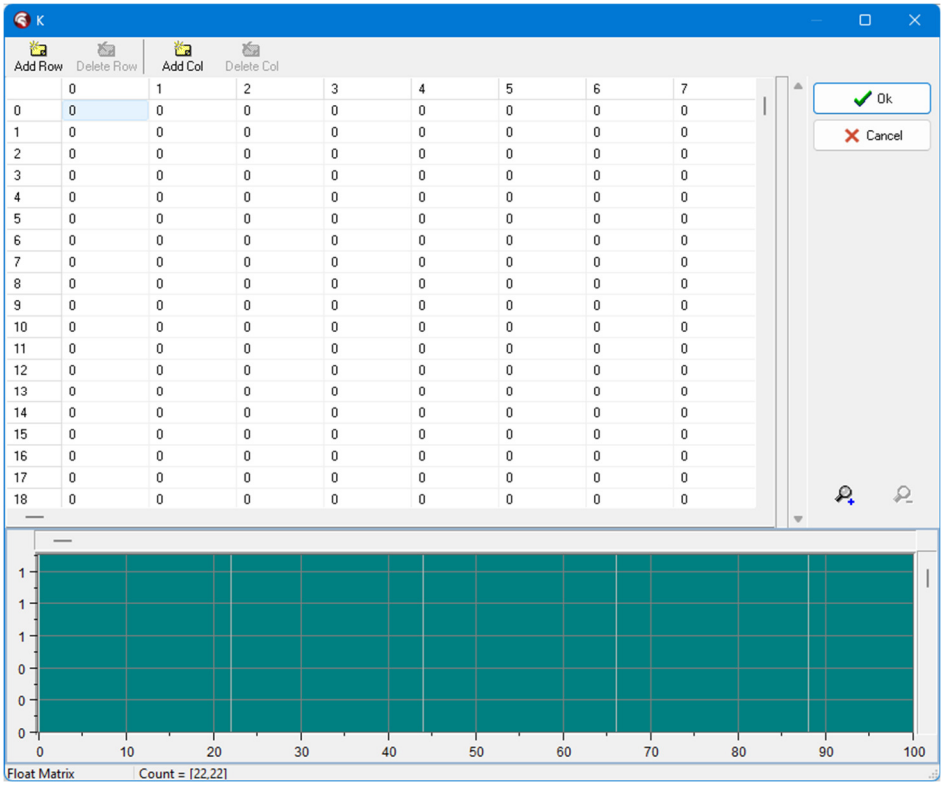
```
procedure SetFloatValueAt(Row, Col: LongWord; Value: Extended);
procedure Sort;
procedure Clear;
procedure Load;
property Data[I: LongWord]: Double read GetData write SetData; default;
property DataIm[I: LongWord]: Double read GetDataIm write SetDataIm;
property SortIx[I: Integer]: Integer read GetSortIx;
```

2.4.4 TFloatMatrix = class(TCustomMatrix)

The TFloatMatrix component properties:



It is possible to modify the matrix content and to approach its content in three ways. The first option is to edit the content using the editing window:

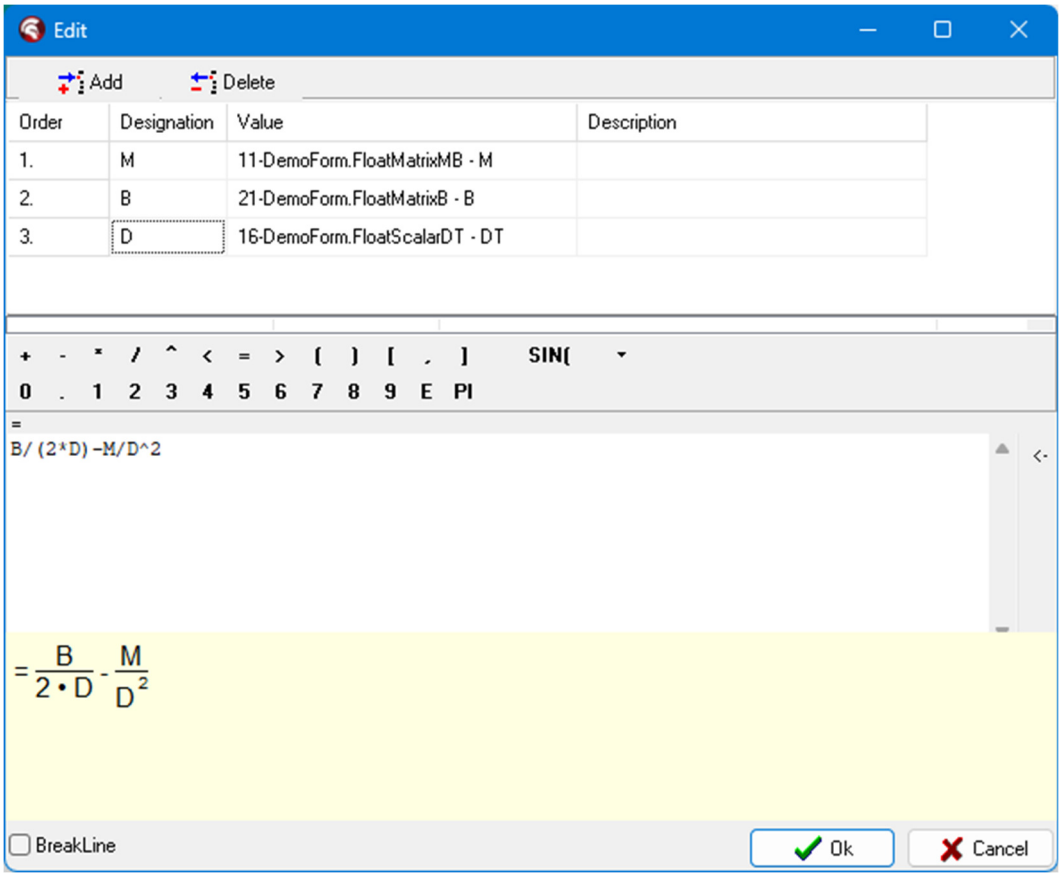


It is necessary to insert the value “True” into the property “**SaveData**” to edit and save the content into the *.dfm file.

The second option is to insert the content / to modify the content and to approach the content in the source code programme using these procedures and properties:

```
procedure Clear;
procedure CalcEquation;
procedure SetSubMatrix(I, J: Integer; SubMatrix: TSingle2D);
procedure GetComplexSubMatrix(I, J: Integer; var SubMatrix: TComplex2D);
procedure SetComplexSubMatrix(I, J: Integer; SubMatrix: TComplex2D);
procedure SetComplexSubVector(I, J: Integer; SubVector: TComplex1D);
procedure AddSubMatrix(I, J: Integer; SubMatrix: TSingle2D);
procedure AddComplexSubMatrix(I, J: Integer; SubMatrix: TComplex2D);
procedure CopyDataFrom(Source: TCustomMatrix);
procedure SetRows(Rows: LongWord; IsEnd: Boolean = False);
property Data[I, J: LongWord]: Single read GetData write SetData; default;
property DataIm[I, J: LongWord]: Single read GetDataIm write SetDataIm;
```

The third option is to calculate the content using the “**Equation**” property:



Then, it is necessary to call the procedure **`CalcEquation`** in the source code.

The TFloatMatrix component has these events:

`property OnSetVarValues: TSetVarValues = = procedure (Sender: TObject; I, J: Integer; var VarValues: TExtended1D) of Object;` - is executed when changing VarValues[0] = T (time).

`property OnGetValue: TCustomFloatVectorGetValue = procedure (Sender: TObject; I: Integer; var V: TComplex) of Object;` - is executed from the procedure `Load`.

`property OnAfterCalc: TNotifyEvent;` - is executed from the procedure `EquationCalc`.

2.5 TPropertyTreeView = class(TCustomTreeView)

TPropertyTreeView component enables to edit the properties of objects derived from TPersistent in the running application. It is possible to reduce significantly the number of necessary dialog windows using the component.

It is imperative that the window in which the TPropertyTreeView component is used is the TDsgnForm child. Just use the DsgnForms_u unit and modify the window definition.

```
type
  TDemoForm = class(TDsgnForm)
```

If the TPropertyTreeView component is used in Frame, it is necessary that Frame is the child of TBaseFrame from the BaseFrame_u unit.

```
type
  TMyFrame = class(TBaseFrame)
```

The selection of objects for the TPropTreeView component:

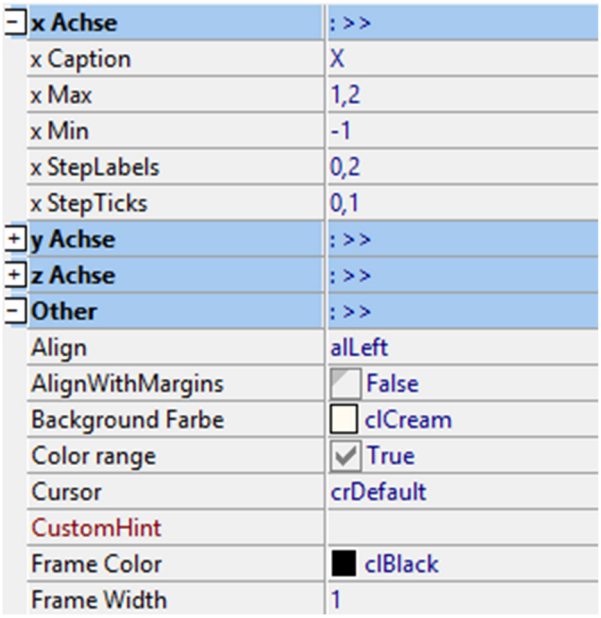
```
procedure TDemoForm.ListBox1Click(Sender: TObject);
var
  I    : Integer;
  AList: TList;
begin
  if ListBox1.ItemIndex = -1 then
    Exit;
  if ListBox1.SelCount > 1 then
  begin
    AList := TList.Create;
    try
      for I := 0 to ListBox1.Count - 1 do
        if ListBox1.Selected[I] then
          AList.Add(ListBox1.Items.Objects[I]);
      PropertyTreeView1.Components := AList;
    finally
      AList.Free;
    end;
  end
  else
    PropertyTreeView1.Persistent :=
TPersistent(ListBox1.Items.Objects[ListBox1.ItemIndex]);
end;
```

The TPropertyTreeView enables sorting properties into categories.

If categories for an object should be defined, it is necessary that it is a TInfoPersistent child from the Info_u unit.

If categories for a component should be defined, it is necessary that it is a TInfoComponent child from the Info_u unit.

If categories for control should be defined, it is necessary that it is a TInfoCustomControl child from the Info_u unit.



It is necessary to override these methods in the TInfoPersistent, or TInfoComponent, or TInfoCustomControl childs:

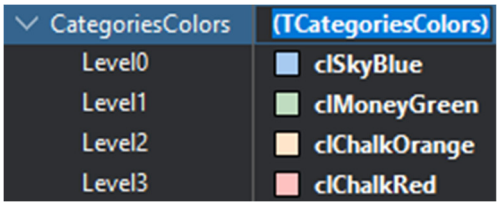
```
  protected
    class procedure BuildCategory(List: TCategoryList); virtual;
  public
    class function CategoryList: TCategoryList; virtual;
```

For example:

```
type
TMyControl = class(TInfoCustomControl)
  protected
    class procedure BuildCategory(List: TCategoryList); override;
  public
    class function CategoryList: TCategoryList; override;
end;

implementation

var
  CategoryList: TCategoryList;

class procedure TMyControl.BuildCategory(List: TCategoryList);
var
  Category: TCategory;  {Create(AName: String; AExpanded: Boolean)}
begin
  Category := TCategory.Create('x Achse', False);
  List.Add(Category);
  Category.AddProperty('x_Min');
  Category.AddProperty('x_Max');
  Category.AddProperty('x_StepTicks');
  Category.AddProperty('x_StepLabels');
  Category.AddProperty('x_Caption');
  Category := TCategory.Create('y Achse', False);
  List.Add(Category);
  Category.AddProperty('y_Min');
  Category.AddProperty('y_Max');
  Category.AddProperty('y_StepTicks');
  Category.AddProperty('y_StepLabels');
  Category.AddProperty('y_Caption');
  Category := TCategory.Create('z Achse', False);
```

```
  List.Add(Category);
  Category.AddProperty('z_Min');
  Category.AddProperty('z_Max');
  Category.AddProperty('z_StepTicks');
  Category.AddProperty('z_StepLabels');
  Category.AddProperty('z_Caption');
end;

class function TMyControl.CategoryList: TCategoryList;
begin
  Result := CategoryList ;
end;

initialization
  CategoryList := TCategoryList.Create;
  TMyControl.BuildCategory(CategoryList);
finalization
  FreeAndNil(CategoryList);
end.
```

It is possible to determine the category color according to the level using the **CategoriesColors** property.



The TPropertyTreeView component has these events:

```
    property OnCompare;
    property OnComponentNameChange: TNotifyEvent;
    property OnSelectNode: TTVChangedEvent
    property OnBeforeSetValue: TSetValueProc;
    property OnSetValue: TSetValueProc;
    property OnFontProperties: TFontPropProc;
```

The TSetValueProc definition:

```
TSetValueProc = procedure (Sender: TObject; PropertyEditor: IProperty; Level: Integer) of
Object;
```

The TFontPropProc definition:

```
TFontPropProc = procedure (Sender: TObject; PE: IProperty; Level: Integer; Name: String;
                           var FontColor: TColor; var FontStyle: TFontStyles;
                           var NewName: String) of Object;
```

This event enables to change FontColor, FontStyle and to change the property name.

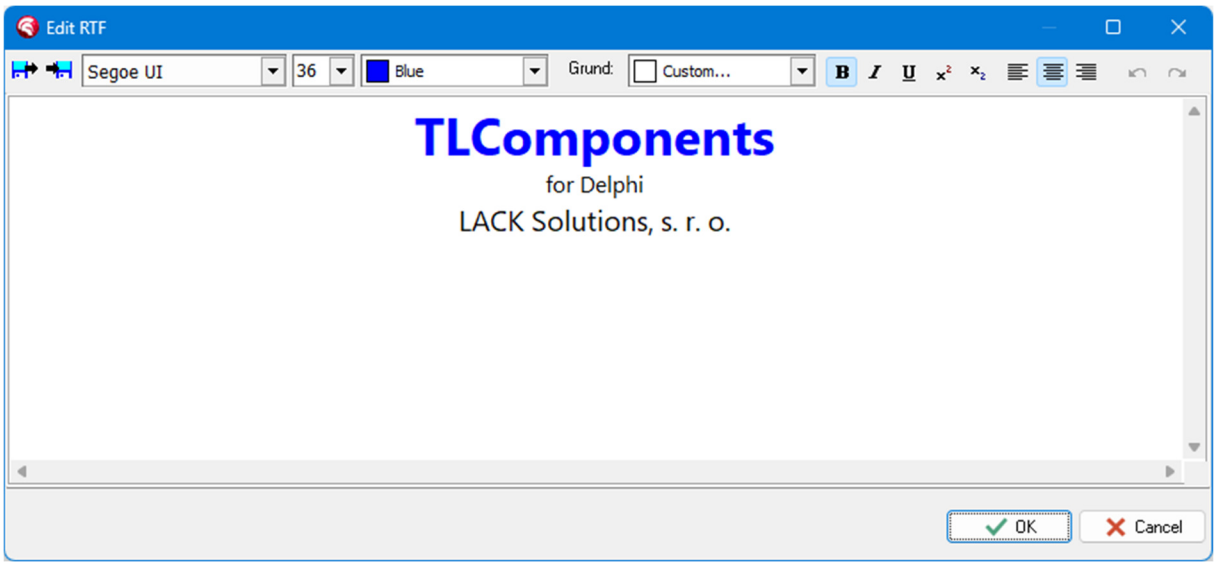
2.6 TLRichEdit = class(TRichEdit)

The TLRichEdit component is the direct child of the TRichEdit component. These procedures are added:

```
procedure CopyToRichEdit(Dest: TLRichEdit);
procedure CopyFromRichEdit(Source: TLRichEdit);
function CopyToString: AnsiString;
procedure CopyFromString(S: AnsiString);
procedure CopyToStream(Stream: TStream);
procedure CopyFromStream(Stream: TStream);
```

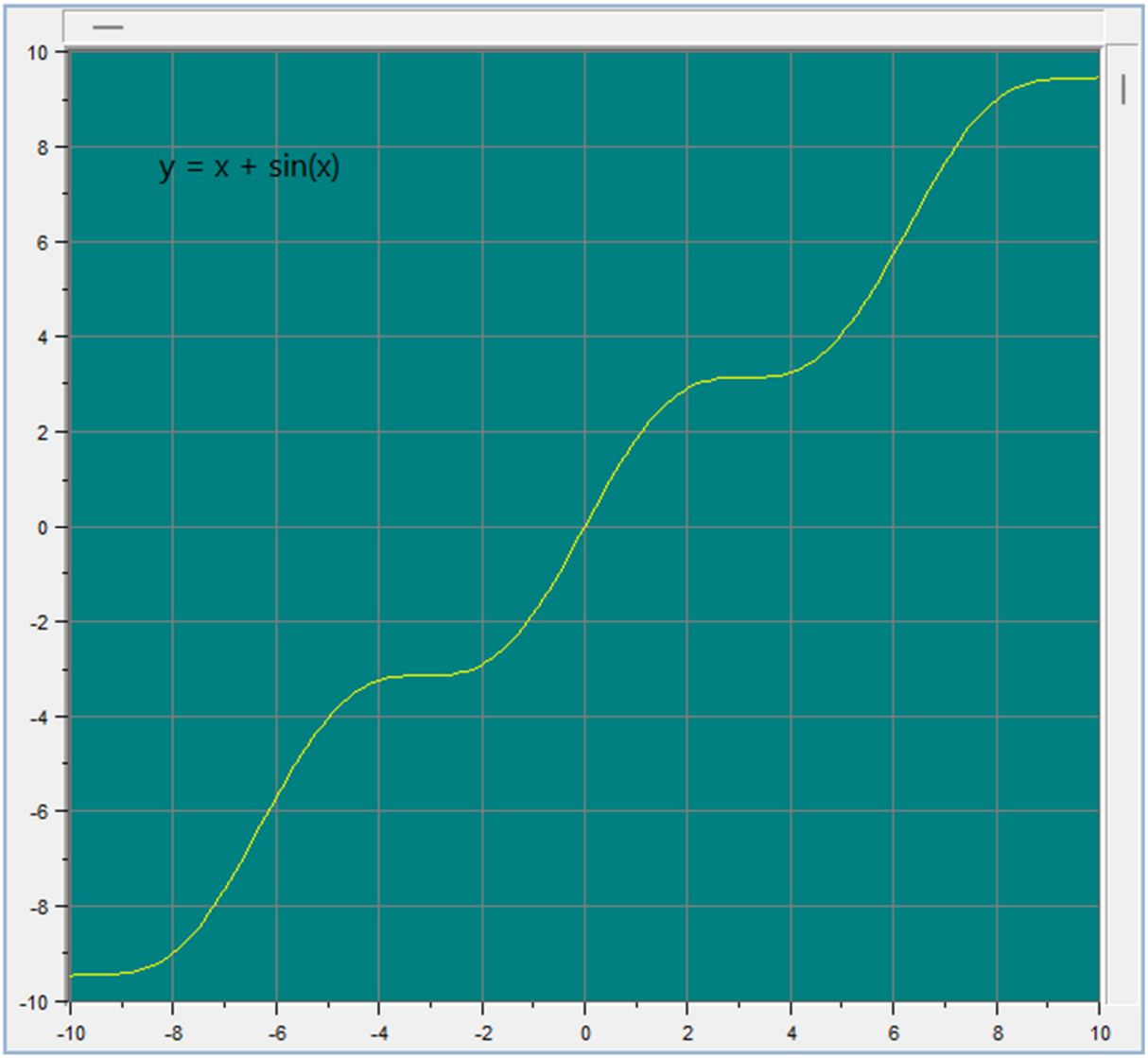
The procedure `CopyToStream` saves into Stream from `Stream.Position`. The procedure `CopyFromStream` loads from Stream from `Stream.Position`.

The editor is added for the TLRichEdit component and **Lines** property:

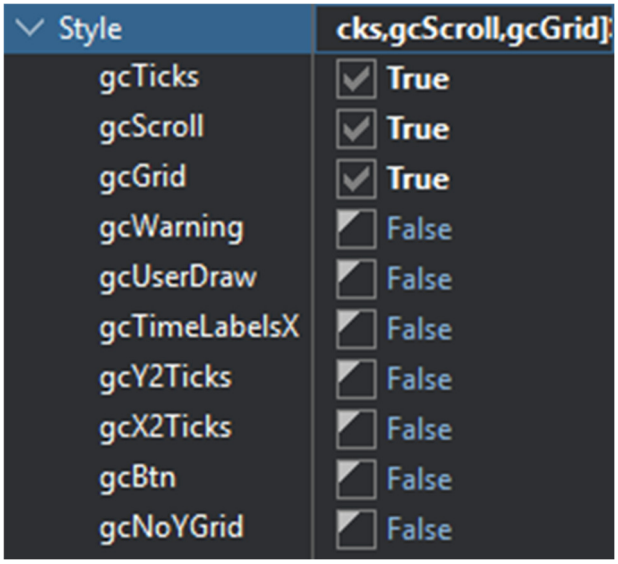


The TLRichEdit component content are saved into a *.dfm file. It is possible to load the *.rtf file into the editor. It is possible to save the editor content into *.rtf. It is possible to copy a formatted text from and into the editor.

2.7 TLGraphControl = class(TInfoCustomControl)



The property **Style** has the principal influence on the appearance and function of the TLGraphControl component.



It is possible to modify the axis range using **Parameters** property and the following dialog.

It is possible to enlarge the graph content with the **ZoonIn** = True property, with the ZoomMode: TZoomMode = (zmRectangle, zmHorizontal, zmVertical) option.

The TLGraphControl component has these added events:

```
    property OnKeyDown: TKeyEvent;
    property OnSetFocus: TNotifyEvent;
    property OnBtnClick: TNotifyEvent;
    property OnGraphMouseLeave: TNotifyEvent;
    property OnXScroll: TScrollEvent;
....property OnYScroll: TScrollEvent;
    property OnCursorPos: TCursorPosEvent = procedure(Sender: TObject; XL, YL: Double) of
Object;
    property OnZoomChange: TZoomChangeEvent = procedure(Sender: TObject; IsZoomIn,
IsZoomed: Boolean) of Object;
    property OnMouseDown: TGMouseEvent = procedure(Sender: TObject; X, Y: Integer) of
Object;
    property OnKeyMouseDown: TGKeyMouseEvent = procedure(Sender: TObject; X, Y, Key:
Integer) of Object;
    property OnPaintGraph: TPaintGraphEvent = procedure(Sender: TObject; Canvas: TCanvas;
PosX, PosY: Integer; MX, MY: Double) of Object;
    property OnPaint: TPaintEvent = procedure(Sender: TObject; Canvas: TCanvas) of Object;
    property OnPaintGrid: TPaintGridEvent = procedure(Sender: TObject; Canvas: TCanvas;
PosX, PosY: Integer; MX, MY, MinX, MinY, MaxX, MaxY, StepX, StepY: Double; R: TRect; var
IsPaint: Boolean) of Object;
    property OnPaintTickX: TPaintTickEvent = procedure(Sender: TObject; Canvas: TCanvas;
Pos: Integer; M, Min, Max, StepT, StepL: Double; R: TRect; var IsPaint: Boolean) of Object;
    property OnPaintTickY: TPaintTickEvent = procedure(Sender: TObject; Canvas: TCanvas;
Pos: Integer; M, Min, Max, StepT, StepL: Double; R: TRect; var IsPaint: Boolean) of Object;
    property OnPaintLabelX: TPaintLabelEvent = procedure(Sender: TObject; Canvas: TCanvas;
Pos: Integer; M, Min, Max, Step: Double; R: TRect; var IsPaint: Boolean) of Object;
    property OnPaintLabelY: TPaintLabelEvent = procedure(Sender: TObject; Canvas: TCanvas;
Pos: Integer; M, Min, Max, Step: Double; R: TRect; var IsPaint: Boolean) of Object;
    property OnPaintTickX2: TPaintTickEvent = procedure(Sender: TObject; Canvas: TCanvas;
Pos: Integer; M, Min, Max, StepT, StepL: Double; R: TRect; var IsPaint: Boolean) of Object;
    property OnPaintTickY2: TPaintTickEvent = procedure(Sender: TObject; Canvas: TCanvas;
Pos: Integer; M, Min, Max, StepT, StepL: Double; R: TRect; var IsPaint: Boolean) of Object;
    property OnPaintLabelX2: TPaintLabelEvent = procedure(Sender: TObject; Canvas: TCanvas;
Pos: Integer; M, Min, Max, Step: Double; R: TRect; var IsPaint: Boolean) of Object;
    property OnPaintLabelY2: TPaintLabelEvent = procedure(Sender: TObject; Canvas: TCanvas;
Pos: Integer; M, Min, Max, Step: Double; R: TRect; var IsPaint: Boolean) of Object;
```

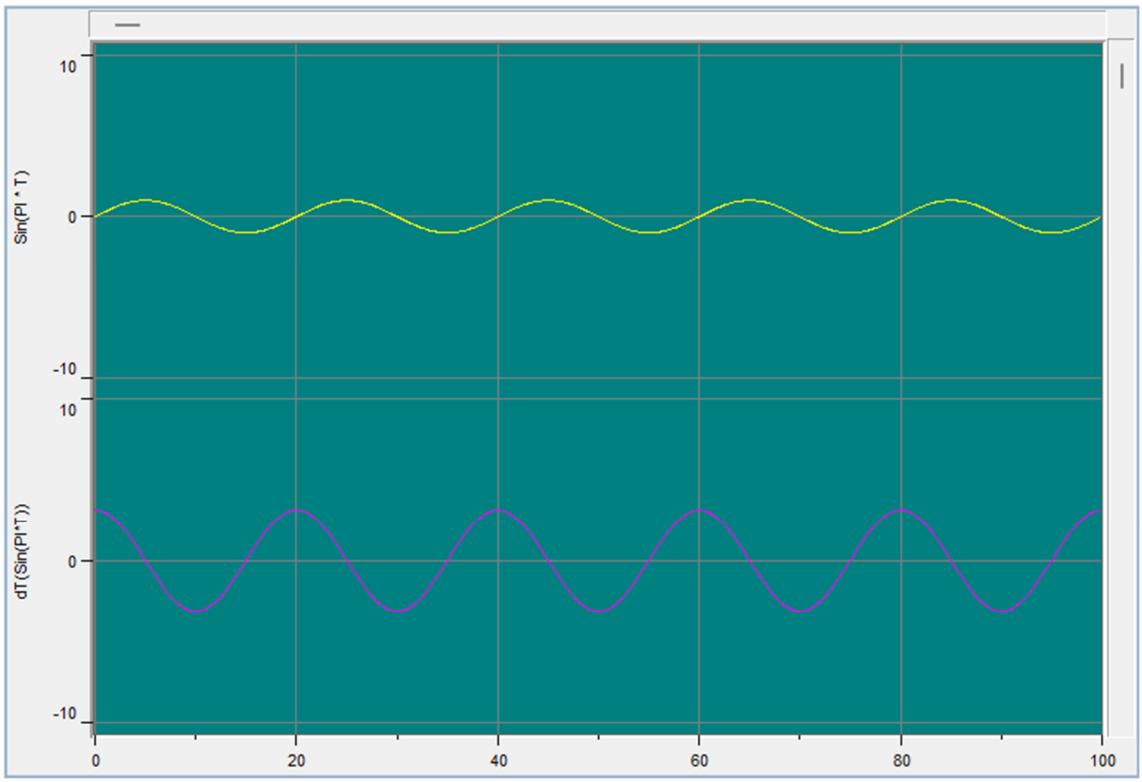
```
    property OnInterval: TIntervalEvent = procedure(Sender: TObject; XL1, XL2: Double) of
Object;
```

OnBtnClick – is executed when clicking the button if Style contains gcBtn
OnCursorPos - is executed when moving the cursor in the graph. XL and YL are logical coordinates.
OnZoomChange – is executed when enlargement is changed. IsZoomIn contains True if further enlargement is possible IsZoomed contains True when the enlargement was applied.
OnPaintGraph – is executed for drawing the content of the graph. It is possible to calculate the coordinates X, Y for drawing from the event parameters in this way:
`X := PosX + Round(XL * MX), Y := PosY – Round(YL * MY)`
Yet, it is possible to use PaintFirstPoint and PaintNextPoint procedures, as in the following example:

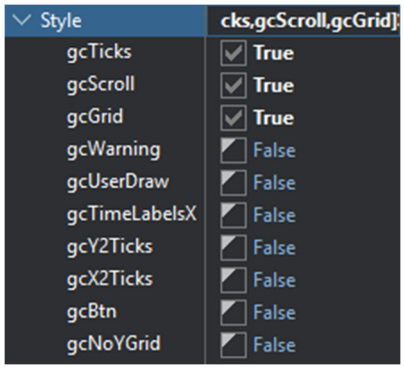
```
procedure TDemoForm.GraphFrame1GraphControl1PaintGraph(Sender: TObject; Canvas: TCanvas;
  PosX, PosY: Integer; MX, MY: Double);
var
  GC  : TLGraphControl;
  X, Y: Single;
begin
  if Equation1.IsError then
    Exit;
  Canvas.font.Height := -18;
  Equation1.Paint(Canvas, 'y = ', 50, 50);
  GC := GraphFrame1.GraphControl1;
  Canvas.Pen.Color := clYellow;
  X := GC.MinXR;
  Y := Equation1.GetValue([X]).Real;
  GC.PaintFirstPoint(Canvas, X, Y); //MoveTo
  repeat
    X := X + 0.1;
    Y := Equation1.GetValue([X]).Real;
    GC.PaintNextPoint(Canvas, X, Y); //LineTo
  until X > GC.MaxXR;
end;
```

OnPaint -it is possible to draw on the base area control using this event.
OnPaintTickX, OnPaintLabelX –user drawing of X axis is possible using this event
OnPaintTickY, OnPaintLabelY – user drawing of Y axis is possible using this event
OnPaintTickX2, OnPaintLabelX2 – user drawing of X2 axis is possible using this event
OnPaintTickY2, OnPaintLabelY2 – user drawing of Y2 axis is possible using this event
It is possible to enable or disable standard drawing using the parameter IsPaint.
OnInterval – is executed for the IntervalIn = True property. XL1 and YL2 are logical coordinates.

2.8 TLGraphControlV = class(TInfoCustomControl)



The property **Style** has the principal influence on the appearance and function of the TLGraphControlV component.

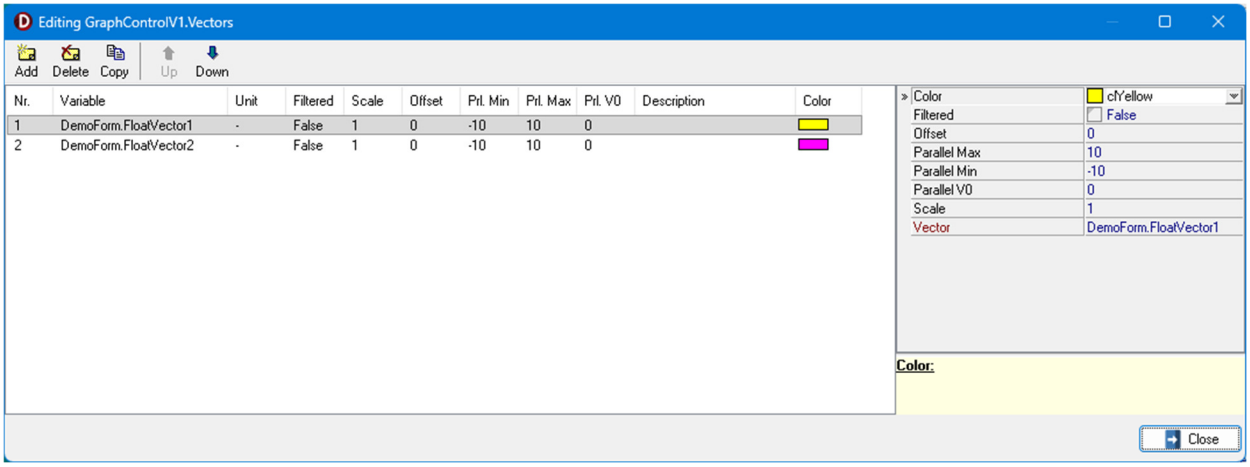


It is possible to modify the axis range using **Parameters** property and the following dialog.

It is possible to enlarge the graph content with the **ZoonIn** = True property, with the **ZoomMode:** TZoomMode = (zmRectangle, zmHorizontal, zmVertical) option.

The TLGraphControlV component is intended to depict the vectors content. If the **Parallel** parameter is set to True, then the vectors are depicted parallel.

The X axis may be defined with **IncX** property or **XVector** property.
The depicted vectors are defined with the **Vectors** property.



If the vector content is gradually calculated, it is possible to send a message about the vector content change with Variables_u.**wm_Update** = 1025 message into the TGraphControlV component.
The procedure is illustrated in the following example:

```
procedure TDemoForm.StartBtnClick(Sender: TObject);
var
  I, J       : Integer;
  MaxU, OMaxU: Double;
  UpdateRec  : TUpdateRec;
begin
  if InProcess then
  begin
    IsStop := True;
    Exit;
  end;
  CloseBtn.Enabled := False;
  StartBtn.Caption := 'Stop';
  try
    InProcess := True;
    IsStop := False;
    UpdateRec.Variable := FloatVectorQE; //Vector
    UpdateRec.Row := 0; //Vector Row
    GraphVFrame2.GraphControlV1.Perform(wm_Update, udClear, NativeInt(@UpdateRec)); //Clear
    FloatVectorQB.CopyDataFrom(FloatVectorQBS);
    FloatVectorOQB.CopyDataFrom(FloatVectorQBS);
    GraphFrame5.GraphControl1.GraphWindow.Refresh;
    TimeLabel.Caption := 'Time = 0';
    Application.ProcessMessages;
    MaxU := 100.0;
    Sleep(250);
    I := 0;
    repeat
      OMaxU := MaxU;
      FloatVectorOOQB.CopyDataFrom(FloatVectorOQB);
      FloatVectorOQB.CopyDataFrom(FloatVectorQB);
```

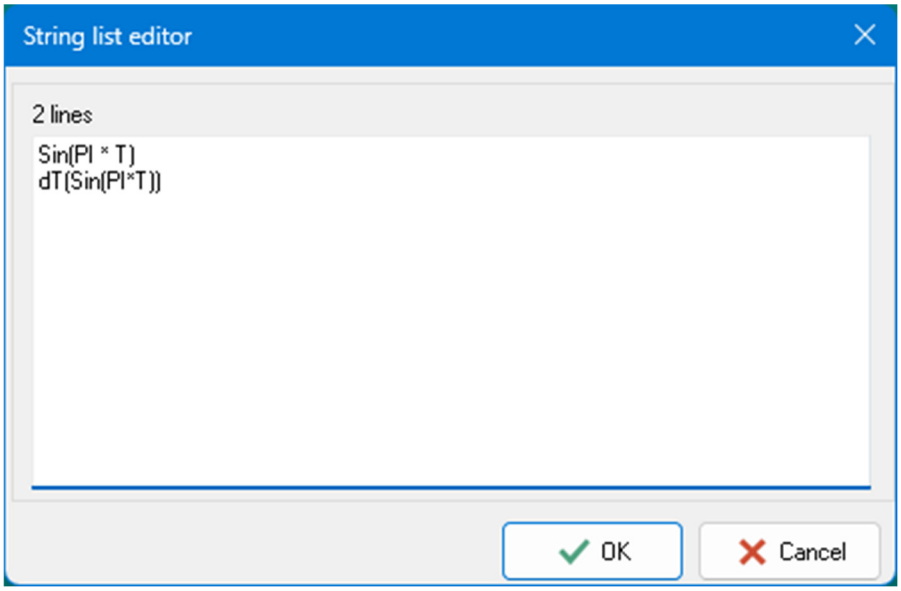
```
      FloatVectorQB.CalcEquation;
      GraphFrame5.GraphControl1.GraphWindow.Refresh;
      Inc(I);
      TimeLabel.Caption := 'Time = ' + FloatToStr(I * FloatScalarDT.Value);
      if I > 1 then
        FloatVectorQE.Count := I + 1;
      FloatVectorQE[I] := FloatVectorQB[FloatVectorQB.Count - 2];
      UpdateRec.Row := I; //Vector row
      GraphVFrame2.GraphControlV1.Perform(wm_Update, udAdd, NativeInt(@UpdateRec));
//AddPoint
      Application.ProcessMessages;
      MaxU := System.Abs(FloatVectorQB[0]);
      for J := 1 to FloatVectorQB.Count - 1 do
        if MaxU < System.Abs(FloatVectorQB[J]) then
          MaxU := System.Abs(FloatVectorQB[J]);
      Sleep(250);
    until IsStop or (I > 400) or ((MaxU < 1.0) and (OMaxU < 1.0));
  finally
    StartBtn.Caption := 'Start';
    CloseBtn.Enabled := True;
    InProcess := False;
  end;
end;
```

The TGraphControlV component contains the same events as the TGraphControl component.

2.9 TGraphVLegend = class(TGraphicsControl)

The TGraphVLegend component is an auxiliary component to the TGraphControlV component.

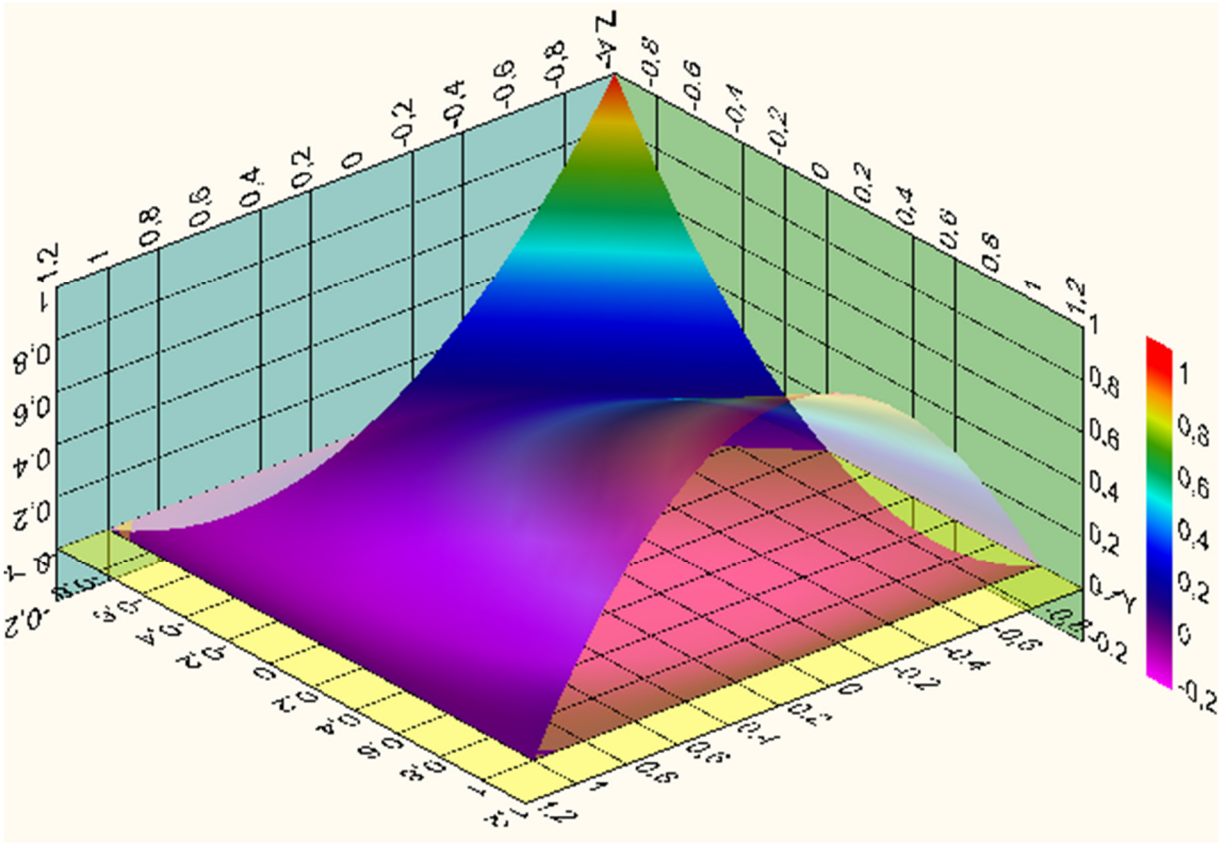
The TGraphControlV component is assigned via the Graph property.
The orientation of component items is selected via the **Orientation** = (loVertical, loHorizontal) property.
It is possible to set the text of items via the **Texts** property.



The width of the items is uniform and it is determined by the **ItemWidth** property.

2.10 TL3DGraph = class(TInfoCustomControl)

This component uses OpenGL for the depiction of a 3D graph.



These properties are for the axis setting:

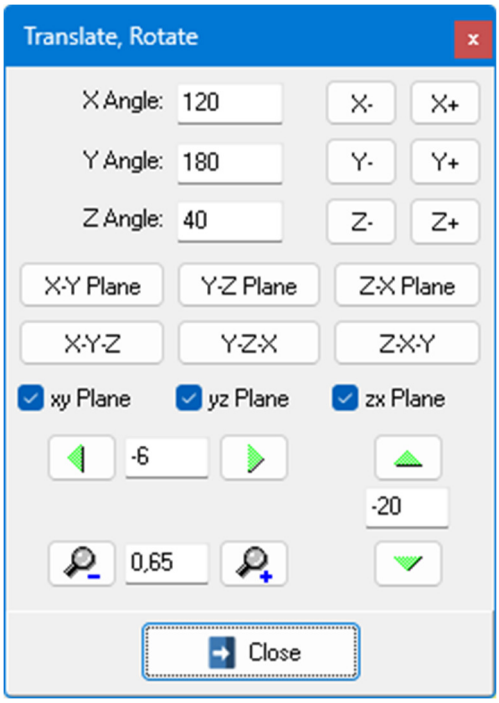
xy_Plane	(TPlane)
x_Caption	X
x_Max	1,2
x_Min	-1
x_StepLabels	0,2
x_StepTicks	0,1
yz_Plane	(TPlane)
y_Caption	Y
y_Max	1,2
y_Min	-1
y_StepLabels	0,2
y_StepTicks	0,1
zx_Plane	(TPlane)
z_Caption	Z
z_Max	1
z_Min	-0,2
z_StepLabels	0,2
z_StepTicks	0,1

TPlane has these properties:

Alpha	0,4
Brush_Color	clYellow
Color	clBlack
Position	0
Visible	True

Position is the position of **Plane** on the orthogonal axis.

The **Translate** and **Rotation** properties are for the setting of the position and orientation of the coordinate system.
It is possible to zoom in and zoom out the coordinate system via the **Scale** property**.**
It is possible to determine the position and orientation of the coordinate system also with the editor of these properties.



It is possible to switch on or switch off the color scale of the z axis via the **Color_range** property.
These properties are for the setting of the lighting:

LightAmbient	0,2; 0,2; 0,2; 0
LightDiffuse	0,4; 0,4; 0,4; 0
LightPosition	0; 500; 500
LightSpecular	0,5; 0,5; 0,5; 0,1

The event **OnBuildGL**: TOnBuildGL = procedure(Sender: TObject; M: Single) of Object; is for drawing into the graph.

It is necessary to multiply all coordinates with the parameter M in the drawing procedure.
It is necessary to use the events from units WinApi.OpenGL and OpenGL_u for drawing.

2.11 TLPageControlV = class(TPageControl)

The TLPageControlV component is the child of the TPageControl component.
This modification of the TPageControl component is usable mainly for vertical usage.
These properties were added in this component:

```
property SelectedColor: TColor;
property TabFont_N: Single;
property TabFont_Bold: Boolean;
```

The **SelectedColor** property enables to determine the color of the selected page.
The **TabFont_N** property enables to change the pages font scale.
The **TabFont_Bold** property enables to make the page Font style Bold.

2.12 TLColorListView = class(TListView)

The TLColorListView component is the child of the TListView component.

The component has one added event:

```
    property OnGetTextColor: TListViewTextColorEvent = procedure (Sender: TObject; Item:
TListItem; SubItem: Integer; var Color, BkColor: TColor) of object;
```

This event enables to change the color and background of the text for ListItem and SubItem

It is necessary to use the event **OnAdvancedCustomDrawSubItem** with a link to an empty procedure together with the event **OnGetTextColor.**

```
procedure TMyForm.ListView1AdvancedCustomDrawSubItem(Sender: TCustomListView; Item:
TListItem; SubItem: Integer;
  State: TCustomDrawState; Stage: TCustomDrawStage; var DefaultDraw: Boolean);
begin
//
end;
```

2.13 TLColorStringGrid = class(TStringGrid)

The TLColorStringGrid component is the child of the je TStringGrid component.

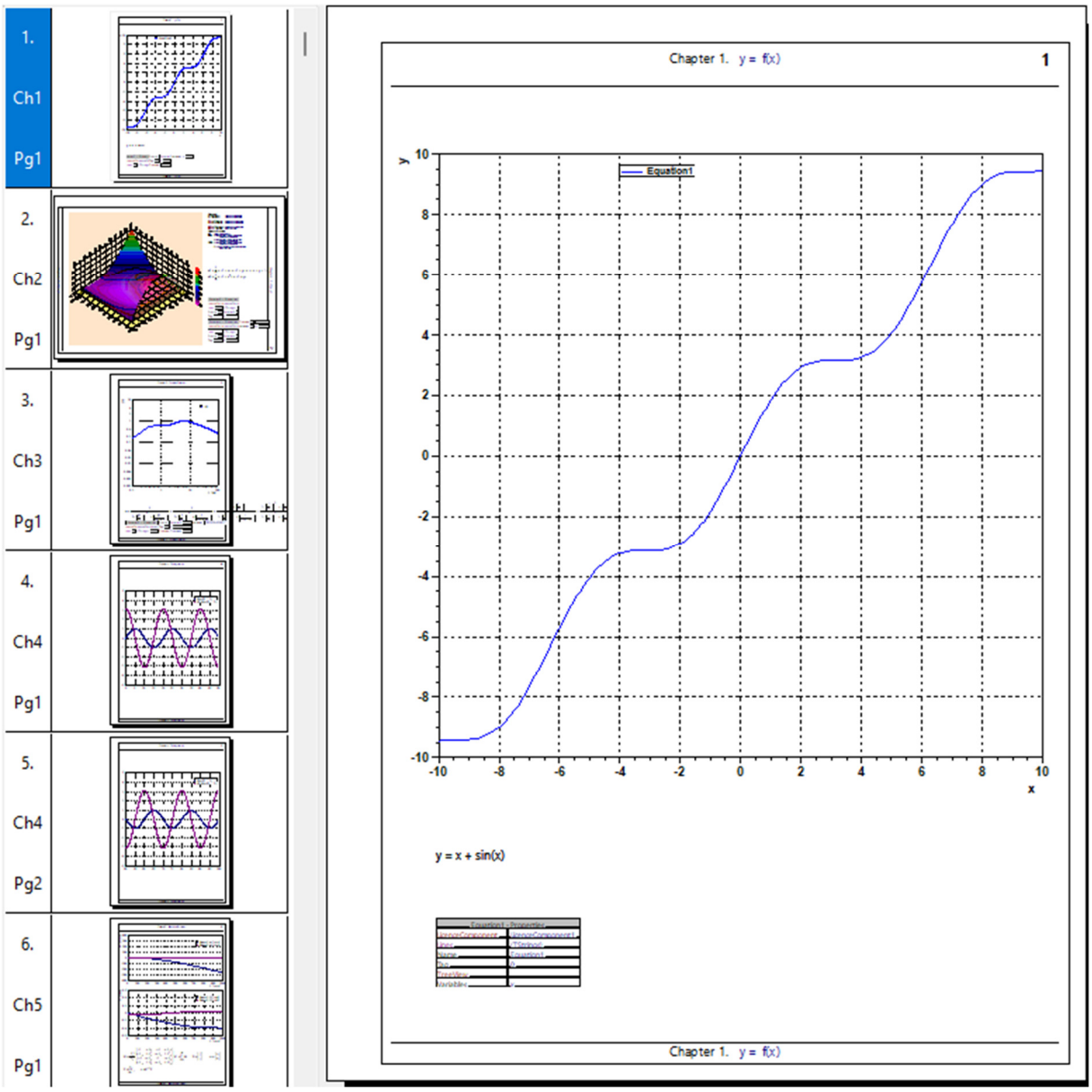
The component has these added events:

```
    property OnGetCellColor: TGetCellColor = procedure (Sender: TObject; ACol, ARow:
LongInt; var Color: TColor) of object;
    property OnGetCellFontProp: TGetCellFontProp = procedure (Sender: TObject; ACol, ARow:
LongInt; var Color: TColor; var Style: TFontStyles) of object;
    property OnDrawCellBG: TDrawCellBGEvent = procedure (Sender: TObject; ARect: TRect;
AColor: TColor; ACol, ARow: Longint; AState: TGridDrawState) of object;
```

Event **OnGetCellColor** enables to determine the color of the cell.
Event **OnGetCellFontProp** enables to determine the color and font style of the cell.
Event **OnDrawCellBG** allows the user to draw on the cell background.

2.14 TLRpt = class(TScrollingWinControl)

The TLRpt component enables to create a report for print.



The component resolution TLRpt is 0.1 mm. It is possible to set the format size via these properties:

FormatHeight	2970
FormatOrientation	foPortrait
FormatSize	fsA4
FormatWidth	2100

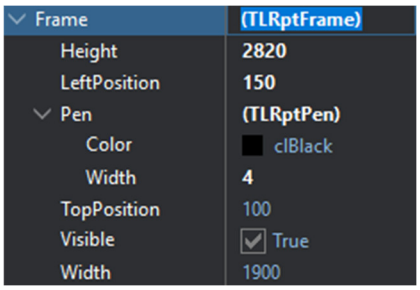
FormatSize has these options:

- fsA0
- fsA1
- fsA2
- fsA3
- fsA4
- fsA5
- fsUser

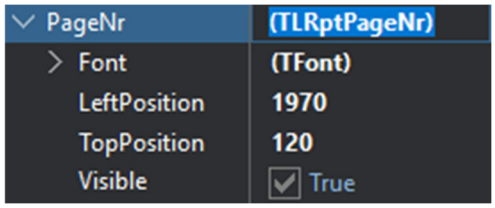
The properties **FormatWidth** and **FormatHeight** are automatically set for from fsA0 to fsA5.

The properties **FormatWidth** and **FormatHeight** can be modified only if **FormatSize** = fsUser.
The property **FormatOrientation** has the options foPortrait or foLandscape.

A drawn frame may be on the pages. It is possible to set drawing of the frame by these properties:



It is possible to modify numbering of pages by these properties:

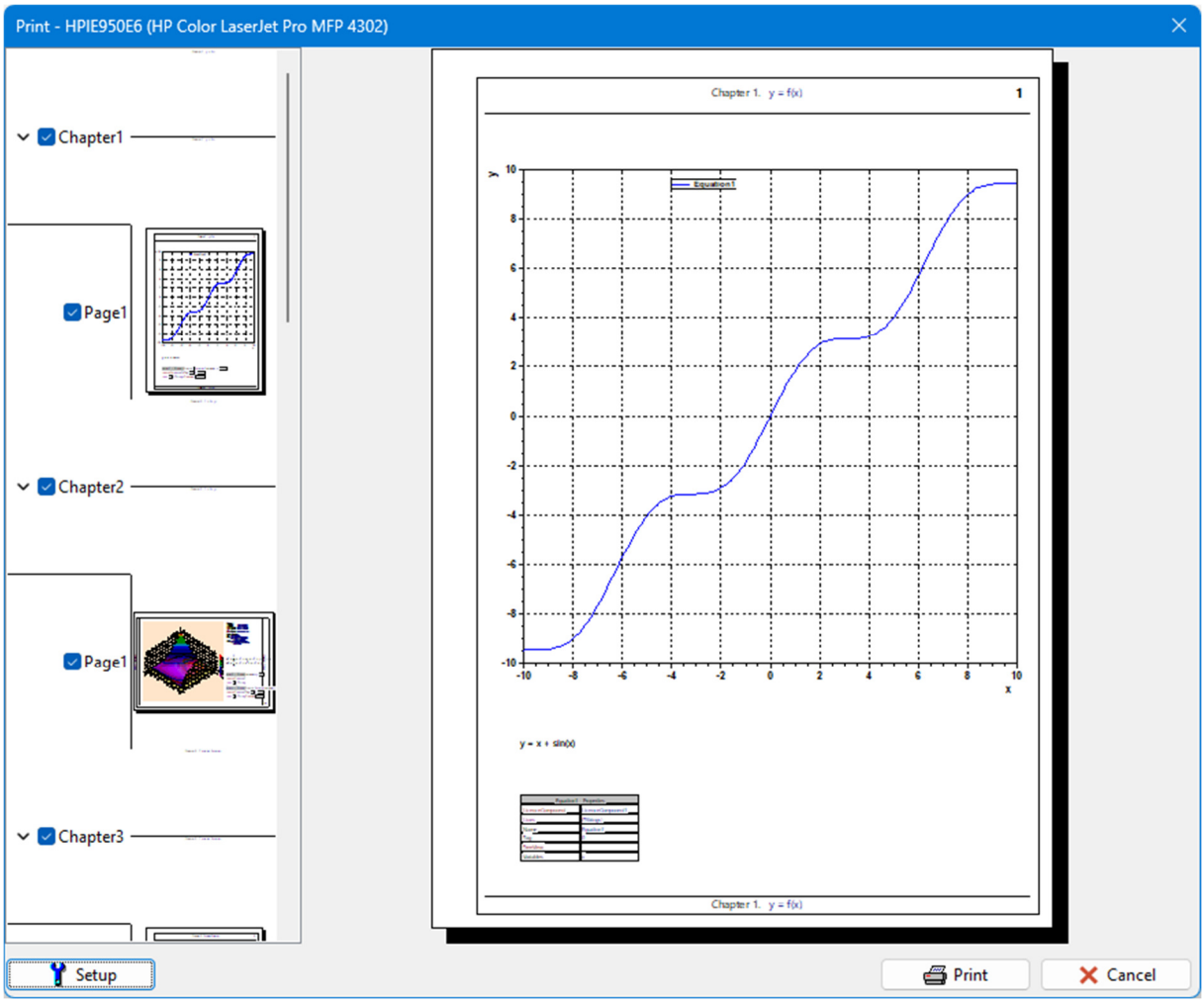


The properties **LeftPosition** and **TopPosition** are valid for the Portrait orientation. They are applied adequately for the Landscape orientation. The property **IsPaintFormat** determines the visibility of the page format. The width of the page shadow is determined by the **ShadowWidth** property.

Important published procedures

ZoomIn – it activates the zoom in regime.
ZoomOut- it returns the previous zoom in.
FullPage(IsRepaint: Boolean) – it depicts the full page.
MoveTopComponent(RptComponent: TLRptComponent) – from a drawing point of view, it moves the RptComponent to the top.
MoveBottomComponent(RptComponent: TLRptComponent) – from a drawing point of view, it moves the RptComponent to the bottom.
MoveUpComponent(RptComponent: TLRptComponent) – from a drawing point of view, it moves the RptComponent up.
MoveDownComponent(RptComponent: TLRptComponent) – from a drawing point of view, it moves the RptComponent down.
RefreshComponent(RptComponent: TLRptComponent; IsErase: Boolean) – it redraws the RptComponent. IsErase determines redrawing of the background.
RptRefresh – it redraws the full background of components.
GlobalPageToChapterAndPage(GlobalPage: Integer; var Chapter, Page: Integer) – based on the global page, it enables to determine the current page and chapter page.
ChapterAndPageToGlobalPage(Chapter, Page: Integer): Integer – this function returns the global page based on chapter and chapter page.
SetChapterAndPage(Chapter, Page: Integer) – it selects a page based on chapter and chapter page.
AddChapter: TLRptChapter – it adds a chapter
ChapterMoveUp(Chapter: TLRptChapter) – it moves a chapter up
ChapterMoveDown(Chapter: TLRptChapter) – it moves a chapter down
ExportWMF(FileName: TFileName; IsRewrite: Boolean) – it esports the current page into a *.wmf or *.emf file type according to the ending of the file. If the ending of the file is *.bmp, *.jpg or *.gif, then it is exported into these formats. IsRewrite determines whether the existing file should be rewritten.
SaveToStream(S: TStream; const Ext, Description: String) – it exports the current page into stream according to the parameter Ext. If Ext is *.wmf, *.emf, *.bmp, *.jpg or *.gif then it is exported into these formats. It exports from Stream.Position.
PrintCurent –it prints the current page with the current setting of the printer.

PrintAll – printing of pages. If the property PrintDialog is assigned, then use this dialog for printing. If the property **PrintDialog** is not assigned, then use the following dialog for printing.



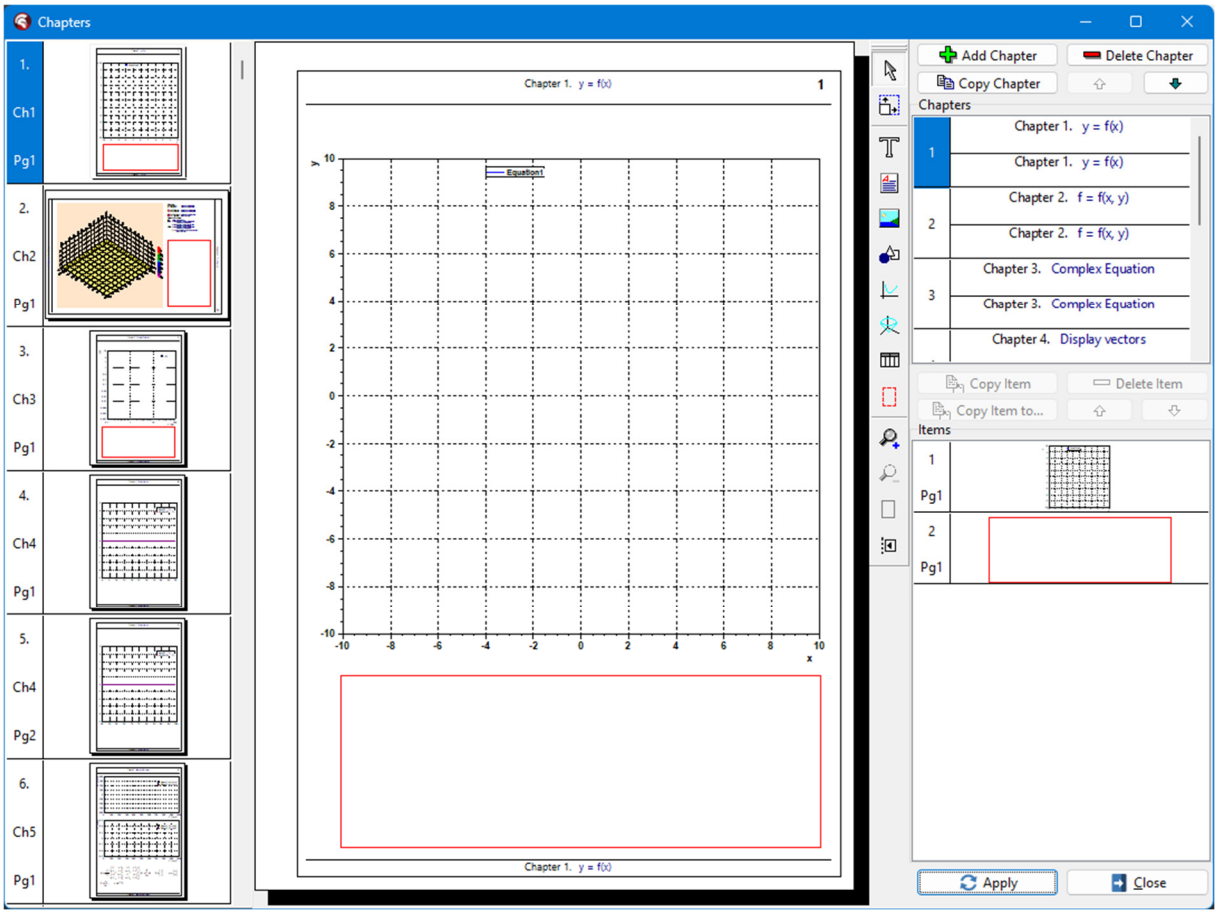
Important events:

OnExportPaint: TNotifyEvent – it is executed when exporting.
OnChangeItemChapter: TNotifyEvent – it is executed when moving an item to another chapter.
OnChangePagesCount : TNotifyEvent – it is executed if the global number of pages was.
OnRefreshPreview: TNotifyEvent – it is executed when there is a need to redraw components TPagesStringGrid or TPagesTreeView.
OnPaint: TPaintEvent = procedure (Sender: TObject; Canvas: TCanvas; R: TRect; Chapter, Page: Integer) of object - it is executed before drawing items in components. It enables additional drawing under items in the component.
OnAddPaint: TPaintEvent = procedure (Sender: TObject; Canvas: TCanvas; R: TRect; Chapter, Page: Integer) of object - it is executed after drawing items in components. It enables additional drawing over items in the component.
OnPostZoom: TZoomEvent = procedure (Sender: TObject; ZoomAction: TZoomAction) of object – it is executed when zooming in / zooming out. ZoomAction = (pzZoomIn, pzZoomOut, pzFullPage, pzNoZoom)
OnPosition: TPositionEvent = procedure (Sender: TObject; P: TPoint) of object – it is executed when moving the mouse on the background of components. The coordinates of the P point are in the logical page coordinates.

OnItemMouseDown: TItemMouseDown = procedure (Sender: TObject; RptItem: TLRptComponent; Button: TMouseButton; Shift: TShiftState; P: TPoint) of object – it is executed when pressing the mouse button on the component. The coordinates of the P point are in the logical page coordinates.
OnItemMove: TItemMouseDown = procedure (Sender: TObject; RptItem: TLRptComponent; Button: TMouseButton; Shift: TShiftState; P: TPoint) of object – it is executed when moving a component across the page. The coordinates of the P point are in the logical page coordinates.
OnInterval: TIntervalEvent = procedure (Sender: TObject; R: TRect) of object – it is executed for the property Status = stXInterval or Status = stYInterval and when choosing page areas with mouse. The coordinate R is in the logical page coordinates.
OnPostCreateComponent: TPostCreateComponent = function(Sender: TObject; RptComponent: TLRptComponent): Boolean of Object – it is executed when adding a new item. If a new item is to be added, its result value must be True.
OnSelectComponent: TSelectEvent = = procedure (Sender: TObject; Component: TLRptComponent; var IsSelect: Boolean) of Object – it is executed when selecting an item. IsSelect determines whether the item should be selected.

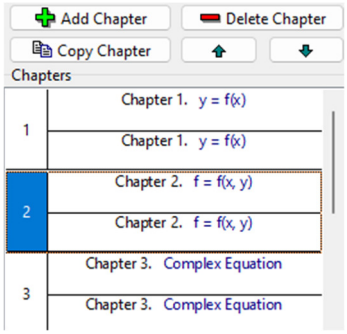
Important properties:

The property **Chapters:** TLRptChaprets is for editing of pages content. The property **Chapters** is edited via this editor:



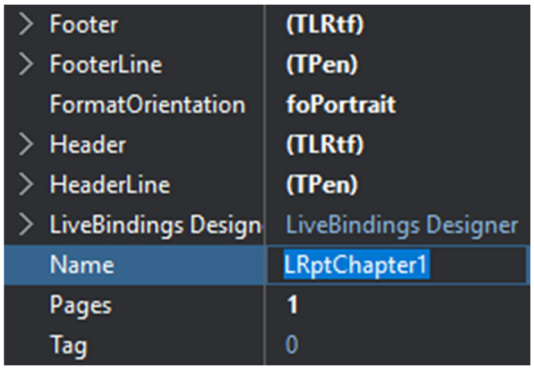
It is necessary to press the button **Apply** after the changes in the editor and the requirement for the changes application.

The following block is used to edit a chapter:



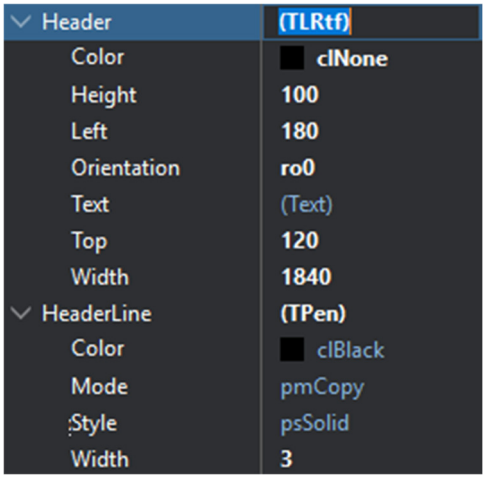
In this block, it is possible to add, delete, copy, change the chapters order and edit a selected chapter.

The chapter has these properties:

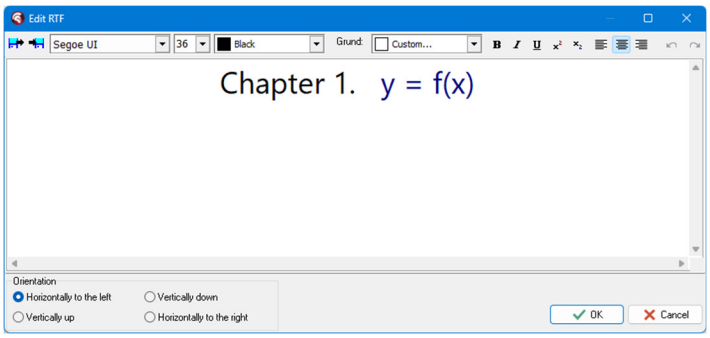


The orientation of the chapter is determined by the **FormatOrientation** property. The properties **Header** and **HeaderLine** are for the header setting. The properties **Footer** and **FooterLine** are for the footer setting. The number of chapter pages is determined by the **Pages** property.

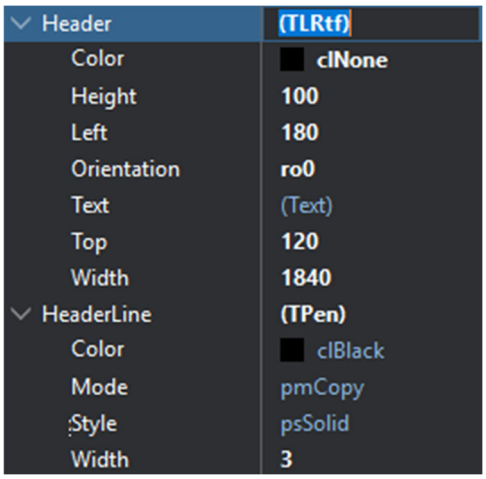
The **Header** and **HeaderLine** properties:



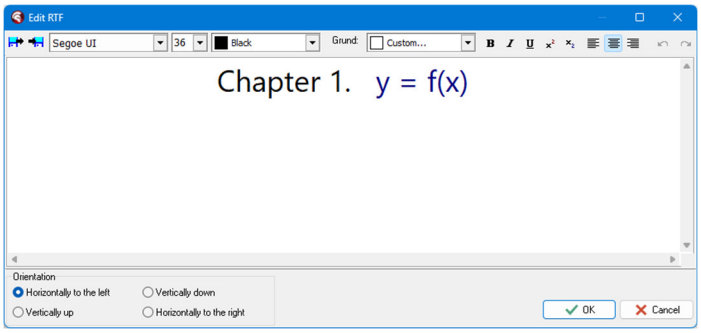
The **Text** property is a formatted Rtf text.



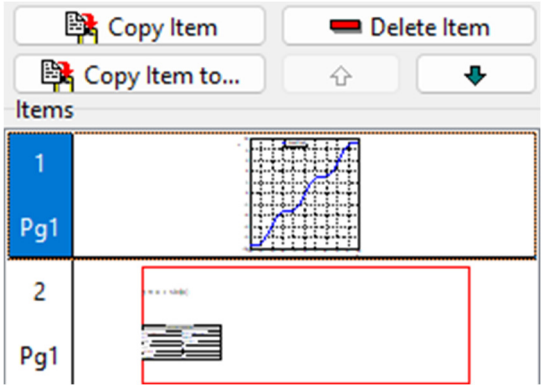
The**Footer** and **FooterLine** properties:



The **Text** property is a formatted Rtf text.



The following block is used to edit items in a chapter:



In this block, it is possible to delete, copy, change the items order and edit a selected item.

In the list of items, there are items of the current chapter.

ToolBar:

- Items selection, after clicking on the item, it is selected in the list of items
- Moving and changing the size of the selected item
- Adding **TLRptText** on the mouse cursor position
- Adding **TLRptRtf** on the mouse cursor position
- Adding **TLRptImage** on the mouse cursor position
- Adding **TLRptShape** on the mouse cursor position
- Adding **TLRptGraph** on the mouse cursor position
- Adding **TLRpt3DGraph** on the mouse cursor position
- Adding **TLRptTable** on the mouse cursor position
- Adding **TLRptPaintBox** on the mouse cursor position
- Zooming in with the mouse
- Zooming out
- Full page
- Switching on the grid for the item movement and size

property **Status** = (stNormal, stZoom, stMove, stExport, stXInterval, stYInterval)

stNormal – selecting items with the mouse (ctrl – reverse order of searching)

stZoom – zooming in with the mouse

stMove -items movement and size change with the mouse

stExport – export of selected items with the mouse into file types *.wmf, *.emf, *.bmp, *,jpg, *.gif

stXInterval, stYInterval – selection of page areas with the mouse. Then, the event **OnInterval** is executed.

property **ExportDialog**: TSaveDialog – it is necessary to assign for the items export into the file types *.wmf, *.emf, *.bmp, *,jpg, *.gif

property **PagesCount**: Integer – a global number of pages of all chapters

property **RptComponents**: TLRptComponents – a list of items

property **SelectedComponent**: TLRptComponent – a selected item

property **MoveGrid**: Boolean – for the True value, the item movement and size is moved with the MoveGridStep step

property **MoveGridStep**: Word- grid step. Standard of 40 (4mm).

2.14.1 TLRptComponent = class(TInfoComponent)

All the items in the TLRpt component must be children of TLRptComponent components.

The TLRptComponent component has these properties:

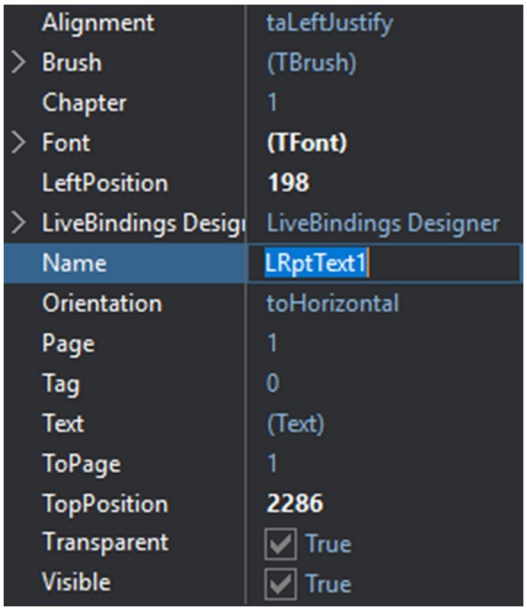
```
    property LeftPosition: Single; - in logical units
    property TopPosition: Single; - in logical units
    property Width: Single; - in logical units
    property Height: Single; - in logical units
    property Visible: Boolean;
    property Chapter: Integer; - a chapter number where the component is situated. If
Chapter= 0, then the component is depicted in all chapters.
    property Page: Integer; - a page number in the chapter. If Page = 0 then the component
is depicted on all pages of the chapters
    property ToPage: Integer; - the component is depicted to this chapter page.
```

The TLRptComponent component has these events:

```
    property OnBeforePaint: TRptComponentPaintEvent = procedure(Sender: TObject; Canvas:
TCanvas; var PaintInfo: TPaintStruct) of Object – it enables drawing before the drawing of
the component itself;
    property OnOutPaint: TRptComponentPaintEvent = procedure(Sender: TObject; Canvas:
TCanvas; var PaintInfo: TPaintStruct) of Object; – it enables drawing after the drawing of
the component itself;
```

2.14.2 TLRptText = class(TLRptComponent)

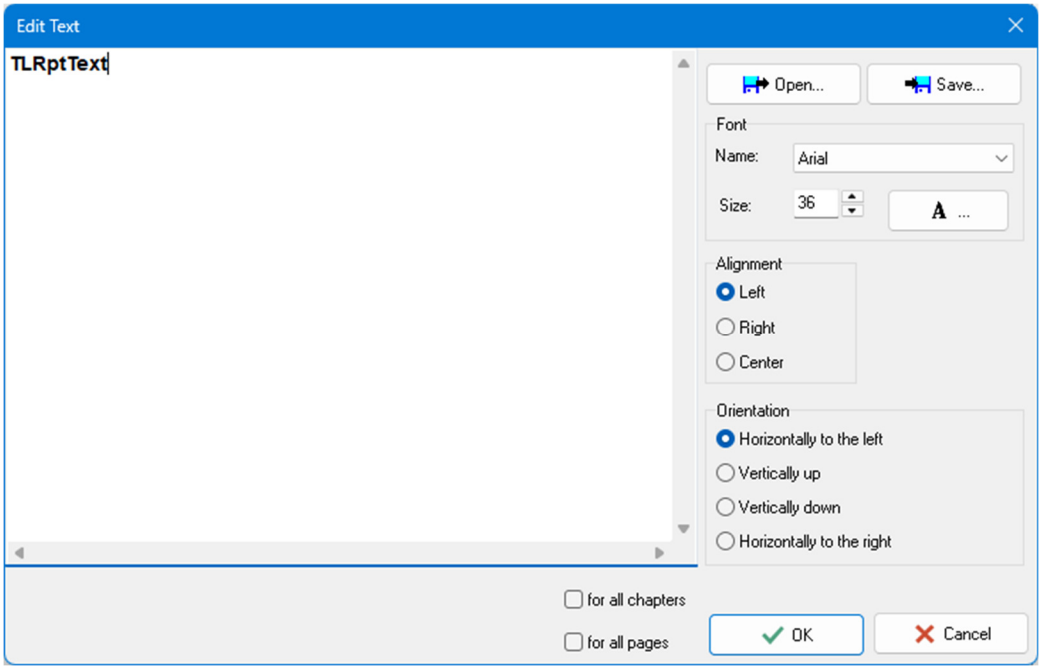
The TLRptText component has these features:



Specific properties of the TLRptText component:

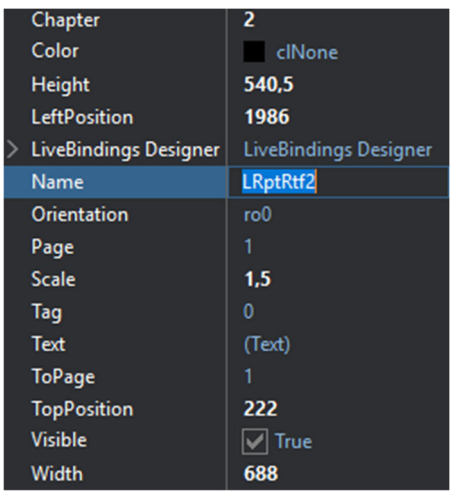
```
property Orientation: TTextOrientation = (toHorizontal, toVertical, toVertical270,
toHorizontal180)

property Transparent: Boolean

property Alignment: TAlignment

property Brush: TBrush

property Font: TFont

property Text: TStringList
```

The following editing window is used for the **Text** property and the component itself:



2.14.3 TLRptRtf = class(TLRptComponent)

The TLRptRtf component has these properties:



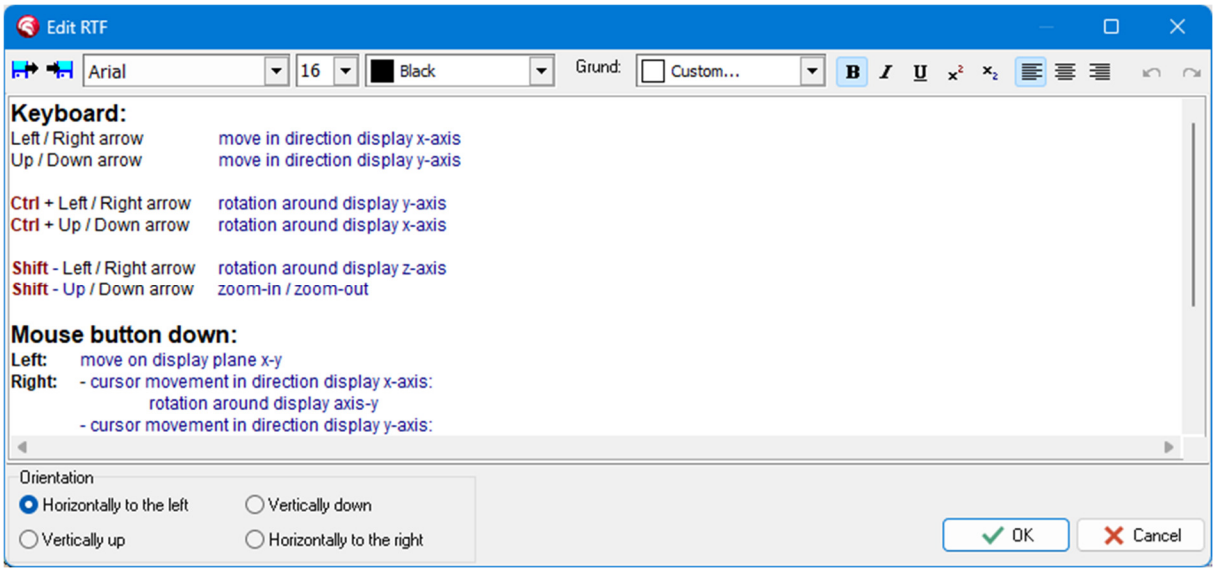
Specific properties of the TLRptRtf component:

```
    property Orientation: TRtfOrientation = (ro0, ro90, ro270, ro180);

    property Color: TColor; - background color. If Color = clNone, then the background is
not drawn.

    property Text: TStrings;

    property Scale: Single;
```

Specific events of the TLRptRtf component:

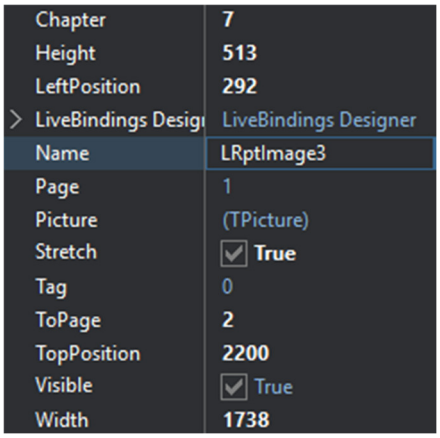
```
    property OnBeforePaint: TRtfPaint = procedure(Sender: TObject; DC: HDC) of Object;

    property OnAfterPaint: TRtfPaint = procedure(Sender: TObject; DC: HDC) of Object;
```

The following editing window is used for the **Text** property and the component itself:

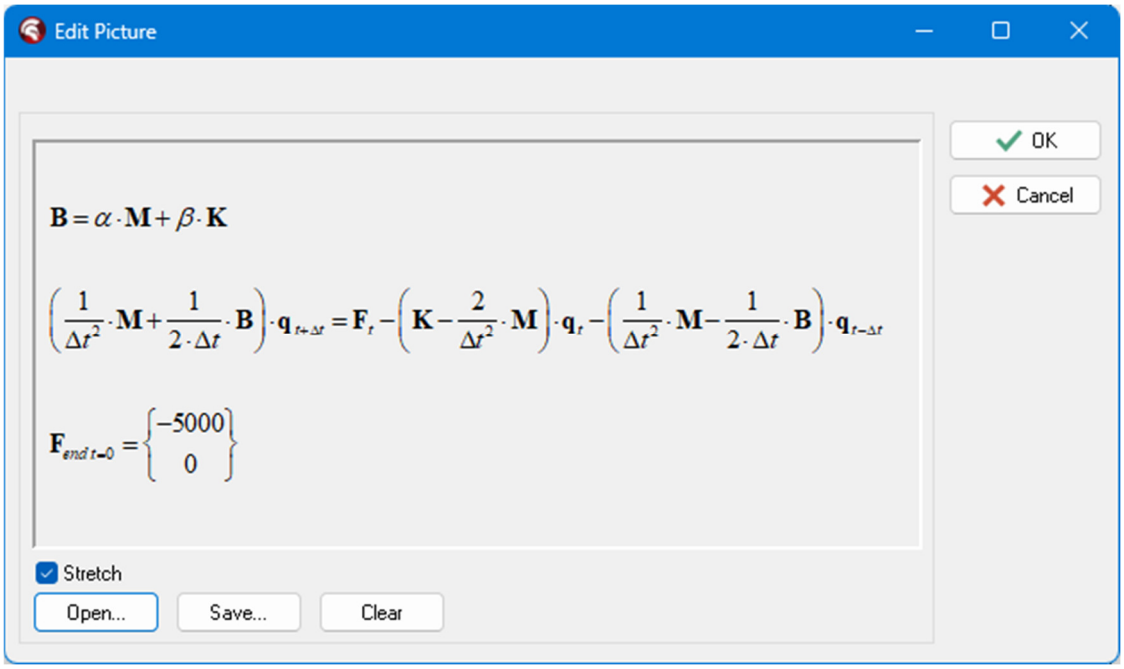


2.14.4 TLRptImage = class(TLRptComponent)

The TLRptImage component has these properties:

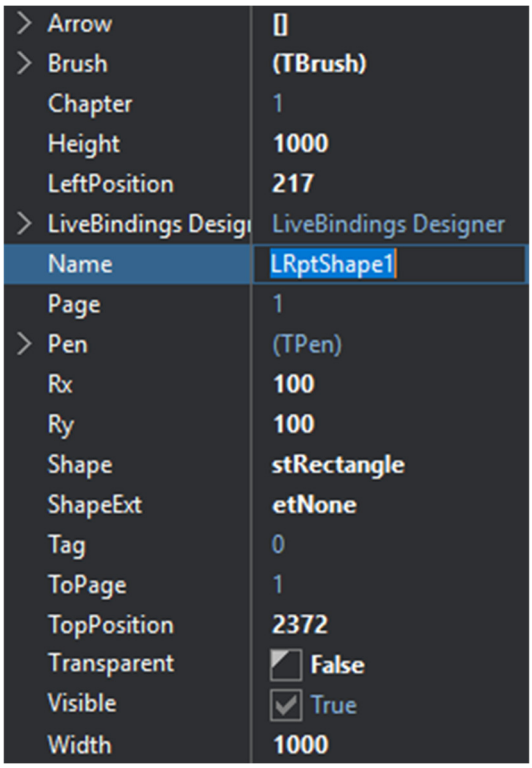


The following editing window is used for the **Picture** property and the component itself:



2.14.5 TLRptShape = class(TLRptComponent)

The TLRptShape component has these properties:



Specific feature of the TLRptShape component:

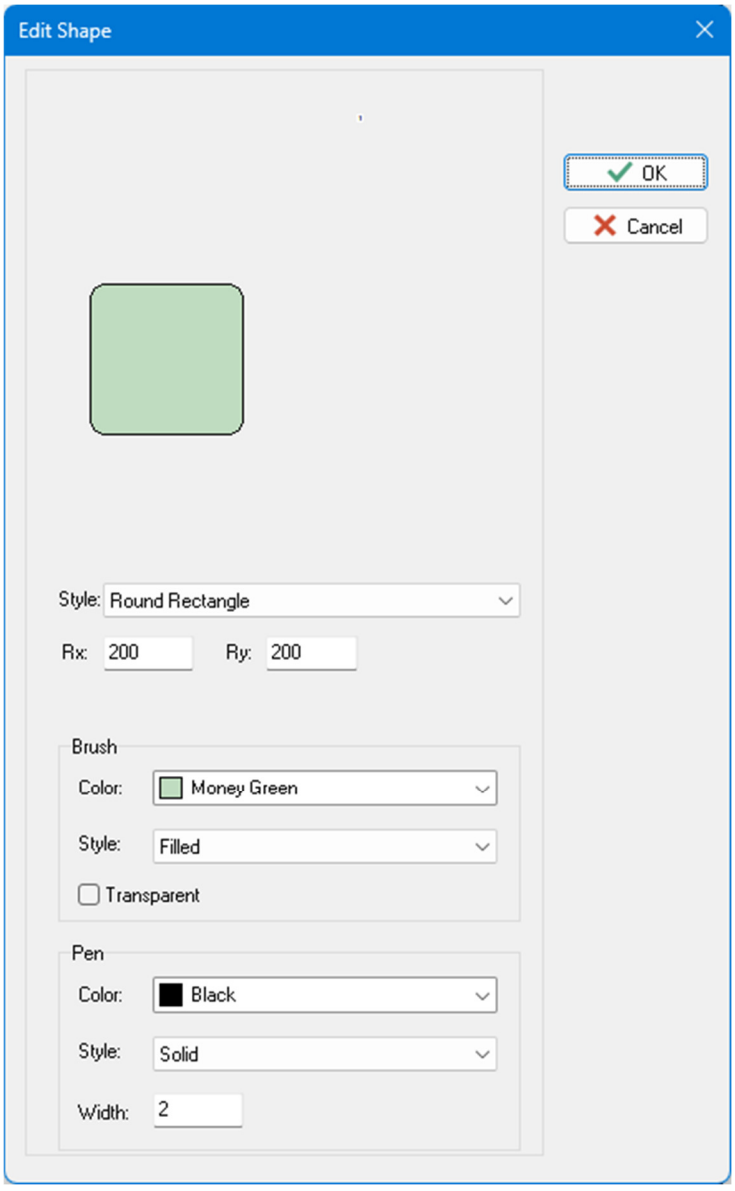
```
property Shape: TShapeType = (stRectangle, stSquare, stRoundRect, stRoundSquare,
stEllipse, stCircle);
```

```
property ShapeExt: TShapeTypeExt = (etNone, etLineH, etLineV, etLineFD, etLineBD);
property Arrow: TArrowTypeSet = set of TArrowType = (atBegin, atEnd);
property Pen: TPen;
property Brush: TBrush;
property Transparent: Boolean;
property Rx: Word; - rounding for rounded shapes
property Ry: Word; - rounding for rounded shapes
```

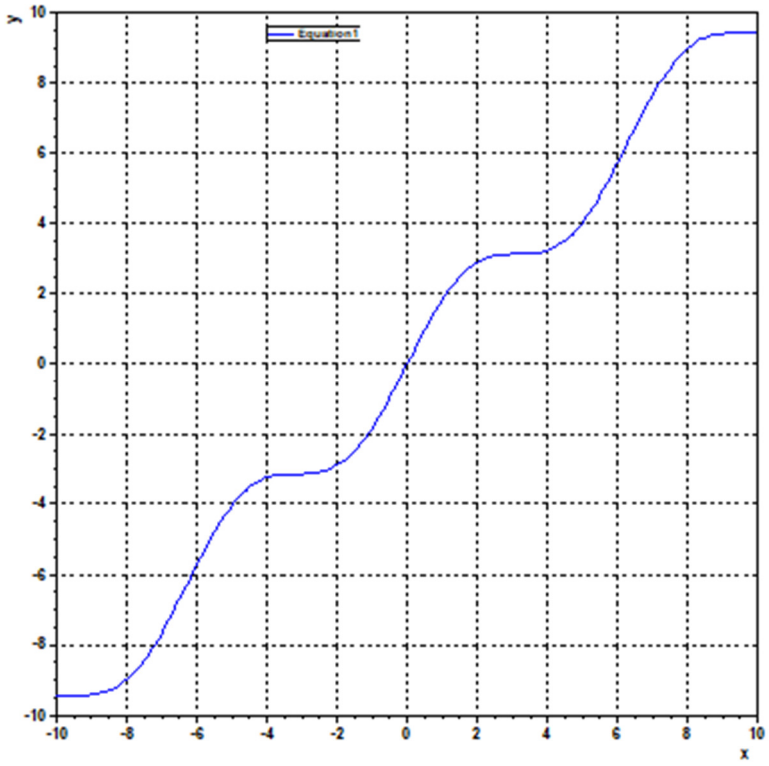
Specific events of the TLRptShape component:

```
property OnPaint: TRptShapePaintEvent = procedure(Sender: TObject; Canvas: TCanvas; Rc: TRect; var PaintInfo: TPaintStruct) of Object;
```

The following editing window is used for TLRptShape component:



2.14.6 TLRptGraph = class(TLRptComponent)



The TLRptGraph component has these properties:

AxisX	(TAxisX)
AxisY	(TAxisY)
BackGround	clNone
Caption	
CaptionPosition	cpTop
Chapter	1
Curves	TCurves(1)
CurvesBackOrder	False
Font	(TFont)
Frame	(TGraphFrame)
Height	1666
LeftPosition	309
Legend	(TRptLegend)
LiveBindings Designer	LiveBindings Designer
MaxX	10
MaxY	10
MinX	-10
MinY	-10
Name	LRptGraph1
Page	1
Stretch	True
Tag	0
ToPage	1
TopPosition	408
Visible	True
Width	1666

Properties **AxisX** and **AxisY**:

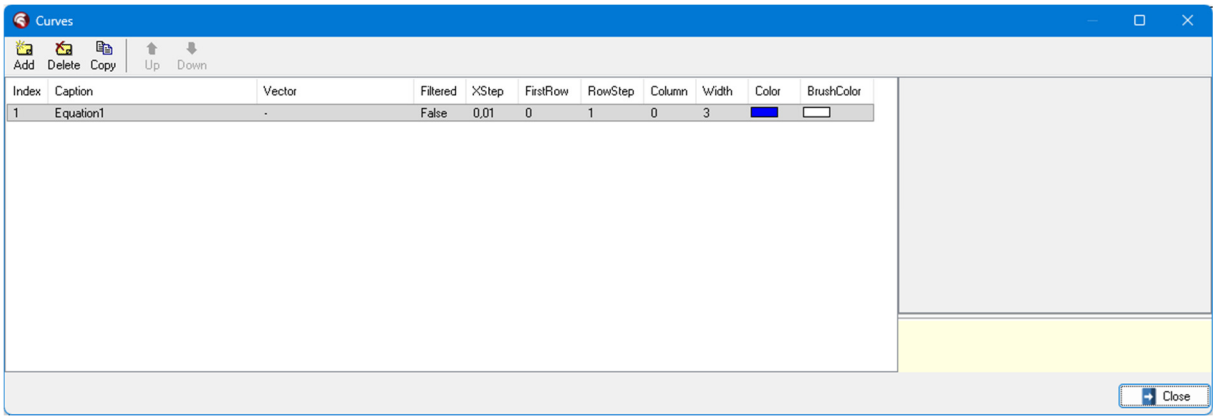
AxisX	(TAxisX)
AppendTxt	
AxisLabel	x
Font	(TFont)
GridVisible	True
LabelFontHeigh	-36
LabelPosition	taRightJustify
LabelsGridLine	(TGridLine)
Logaritmic10	False
Pen	(TLRptPen)
Position	yBottom
StepLabels	2
StepTicks	0,5
TicksGridLine	(TGridLine)
TimeLabels	False
Visible	True

AxisY	(TAxisY)
AppendTxt	
AxisLabel	y
Font	(TFont)
GridVisible	True
IsY2	False
Label2	
LabelFontHeight	-36
LabelPosition	tlTop
LabelsGridLine	(TGridLine)
Logaritmic10	False
Margin	0
Pen	(TLRptPen)
Position	xLeft
StepLabels	2
StepTicks	0,5
TicksGridLine	(TGridLine)
Visible	True
Y2A	0
Y2B	1
Y2Color	clBlack
YColor	clBlack

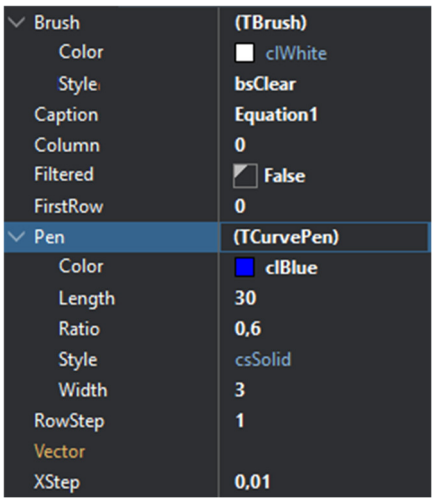
It is possible to determine the graph appearance, the axis range, **AxisX**, **AxisY**, **Curves** and **Legend** via these properties. The axis may be logarithmical if the property **Logaritmic10** = True. If the axis is logarithmical, then the minimum and maximum should be values of logarithms to base 10.

The graph may have also a secondary axis Y on the right side if the property **IsY2** = True. Then the shifting and multiple of secondary axis values is determined by the **Y2A** a **Y2B** properties.

The following editing window is used for the **Curves** property and the component itself:



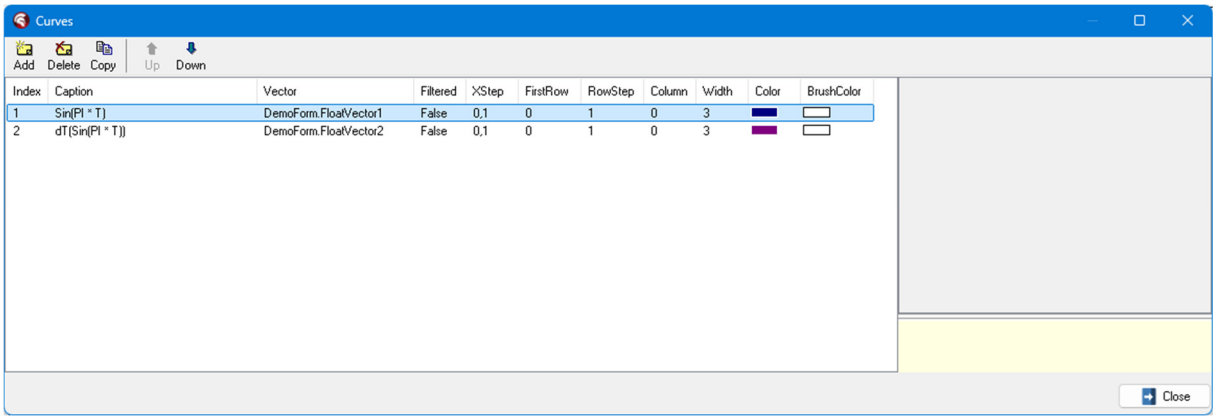
It is possible to add and edit objects **Curve**: TCurve with these properties in this window:



If the property **Vector** = nil, then the properties **Column**, **Filtered**, **FirstRow**, **RowStep** and **XStep** are ignored. The curve is drawn in the **OnPaintCurve** event, as it is illustrated in the following example.

```
procedure TDemoForm.LRptGraph1PaintCurve(Curve: TCurve);
var
  X, Y: Single;
begin
  if Equation1.IsError then
    Exit;
  if Curve.Index = 0 then
  begin
    X := Curve.Graph.MinX;
    Y := Equation1.GetValue([X]).Real;
    Curve.FirstPoint(X, Y);
    repeat
      X := X + 0.1;
      Y := Equation1.GetValue([X]).Real;
      Curve.NextPoint(X, Y);
    until X > Curve.Graph.MaxX;
  end;
end;
```

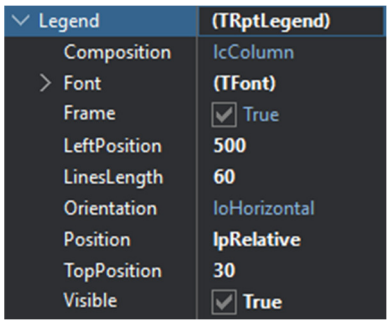
If we want to draw vectors, then the **Curves** property is configurated, for example, like this:



In this case, the **OnPaintCurve** event is not used. The vectors are drawn automatically. If the matrix column content should be drawn, it is necessary to determine the column via the **Column** property.

If the component property is TLRptGraph **ToPage** > **Page**, then the drawing of the vector continues on the following pages.

The graph legend may be configurated via the **Legend**: TRptLegend = class(TInfoPersistent) property with these properties:



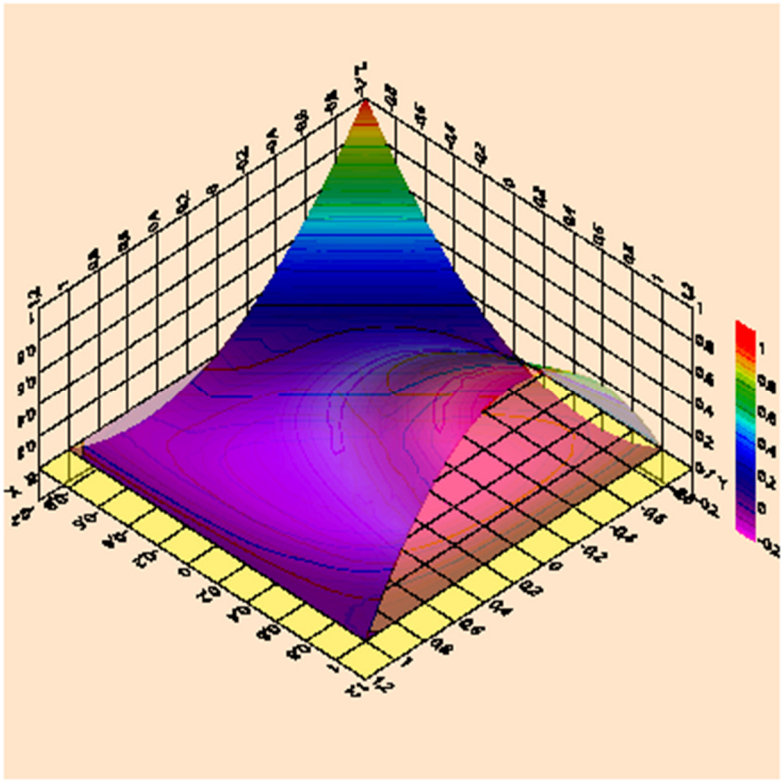
Specific events of the TLRptGraph component:

```
TRptGraphPaintEvent = procedure(Sender: TObject; Page, DC: Integer;
    var PaintInfo: TPaintStruct; PosX, PosY: Integer; MX, MY: Double) of Object;

    property OnPaintBackground: TRptGraphPaintEvent;
    property OnPaint: TRptGraphPaintEvent;
    property OnPaintAxis: TRptGraphPaintEvent;
    property OnBeforePaintAxis: TRptGraphPaintEvent;
    property OnAfterPaintAxis: TRptGraphPaintEvent;
    property OnPaintCurve: TRptGraphPaintCurve: TRptGraphPaintCurve = procedure(Curve:
TCurve) of Object;
    property OnAfterCurvesPaint: TRptGraphPaintEvent;
    property OnCurveProperties: TRptGraphCurveProperties TRptGraphCurveProperties =
procedure(Curve: TCurve; var Caption: String; Pen: TPen; Brush: TBrush) of Object;
```

It is possible to use the events **OnPaintAxis**, **OnBeforePaintAxis**, **OnAfterPaintAxis** when a non-standard axis is required.

It is possible to change **Curve** properties before its drawing via the **OnCurveProperties** event.

2.14.7 TLRpt3DGraph = class(TLRptComponent)



The TLRpt3DGraph component is practically the TL3DGraph component situated in the TLRpt component, which is described in section 2.10.

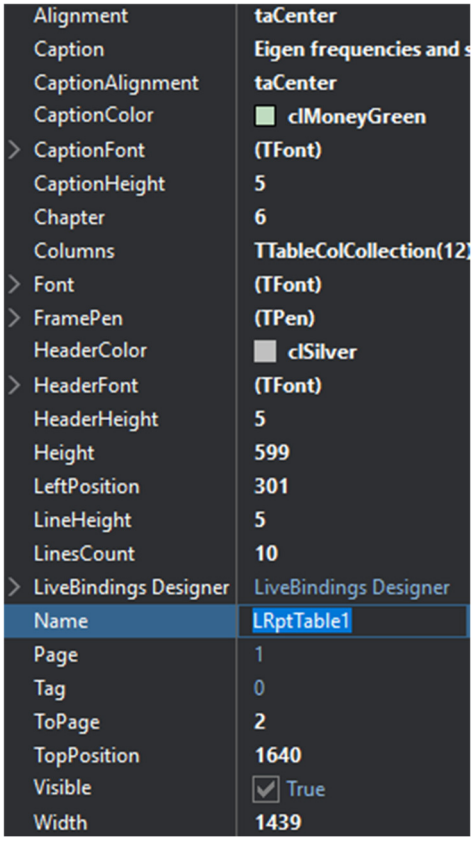
This component has also the **OnBuildGL**: TOnBuildGL = procedure(Sender: TObject; M: Single) of Object event, which is the same as in the TL3DGraph component. It makes it possible to use the same procedure for the drawing of the graphs content TLRpt3DGraph and TL3DGraph.

2.14.8 TLRptTable = class(TLRptComponent)

Eigen frequencies and shapes											
f [Hz]	x=0	x=0,1	x=0,2	x=0,3	x=0,4	x=0,5	x=0,6	x=0,7	x=0,8	x=0,9	x=1
0,503	0	0,017	0,064	0,136	0,23	0,34	0,461	0,591	0,725	0,862	1
3,152	0	-0,093	-0,301	-0,526	-0,683	-0,714	-0,589	-0,317	0,07	0,524	1
8,827	0	0,228	0,605	0,756	0,526	0,02	-0,474	-0,657	-0,395	0,228	1
17,31	0	-0,385	-0,754	-0,434	0,316	0,707	0,327	-0,397	-0,643	-0,052	1
28,659	0	0,537	0,66	-0,211	-0,697	0,001	0,7	0,226	-0,6	-0,294	1
42,934	0	-0,662	-0,338	0,67	0,111	-0,708	0,11	0,677	-0,304	-0,485	1
60,239	0	0,74	-0,102	-0,574	0,632	0,003	-0,634	0,573	0,124	-0,611	1
80,704	0	-0,763	0,5	0,002	-0,509	0,715	-0,5	-0,009	0,518	-0,667	1
104,349	0	0,731	-0,721	0,591	-0,326	-0,012	0,348	-0,605	0,73	-0,65	1
129,721	0	-0,661	0,72	-0,786	0,823	-0,833	0,815	-0,769	0,699	-0,574	1

This component is used for the depiction of tables of values.

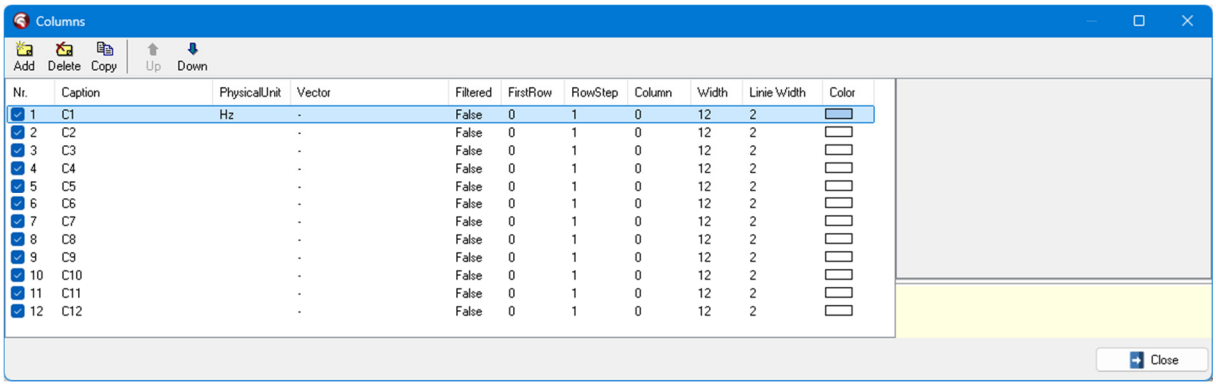
The TLRptTable component has these properties:



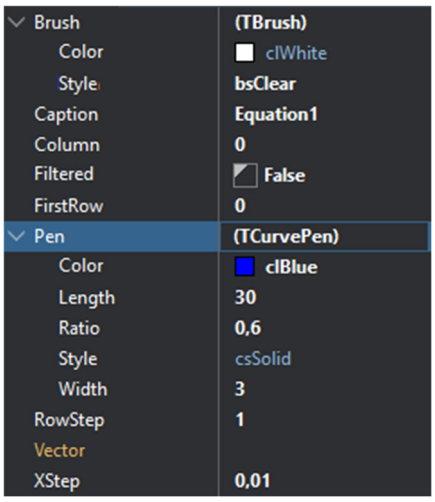
The **LineHeight** property determines the height of the table line. The **LinesCount** property determines the number of table lines on a page. In tis way, a table may be depicted on several pages. The **CaptionHeight** property determines the height of the table caption and the **HeaderHeight** property determines the height of the table header. The property **CaptionAlignment** determines the alignment of the caption, and the property **Alignment** determines the alignment of the table columns.

The **Columns** property determines the table columns and their properties.

The following editing window is used for the **Columns** property and the component itself:



The **Column** object has these properties:



If the property **Vector** is not assigned, then the properties **Column**, **Filtered**, **FirstRow** and **RowStep** and **XStep** are ignored.

It is necessary to determine the table cells via the **OnGetCellValue** event.

Via its assignment, the **Vector** property enables to write out the vector content in the column cells.

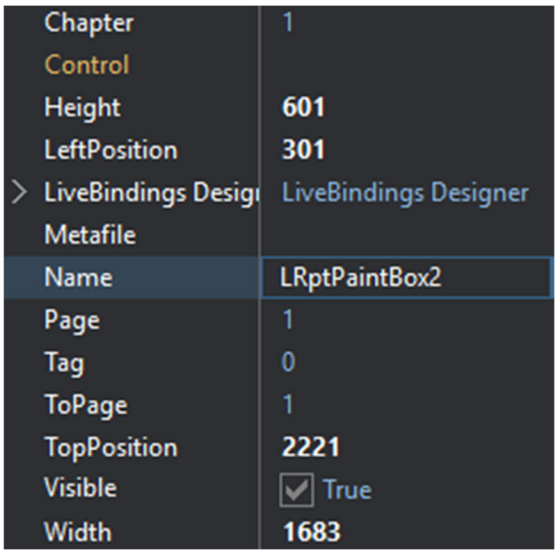
If we want to write out the matrix column content, it is necessary to determine the matrix column with the **Column** property.

Specific events of the TLRptTable component:

```
    property OnGetRowCount: TGetRowCount = procedure(Sender: TObject; var RowCount: Integer)
of Object;

    property OnGetCellValue: TGetCellValue = procedure(Sender: TObject; Col, Row: Integer;
Font: TFont; var BkColor: TColor; var Value: String) of Object;
```

2.14.9 TLRptPaintBox = class(TLRptComponent)

The TLRptPaintBox component has these properties:



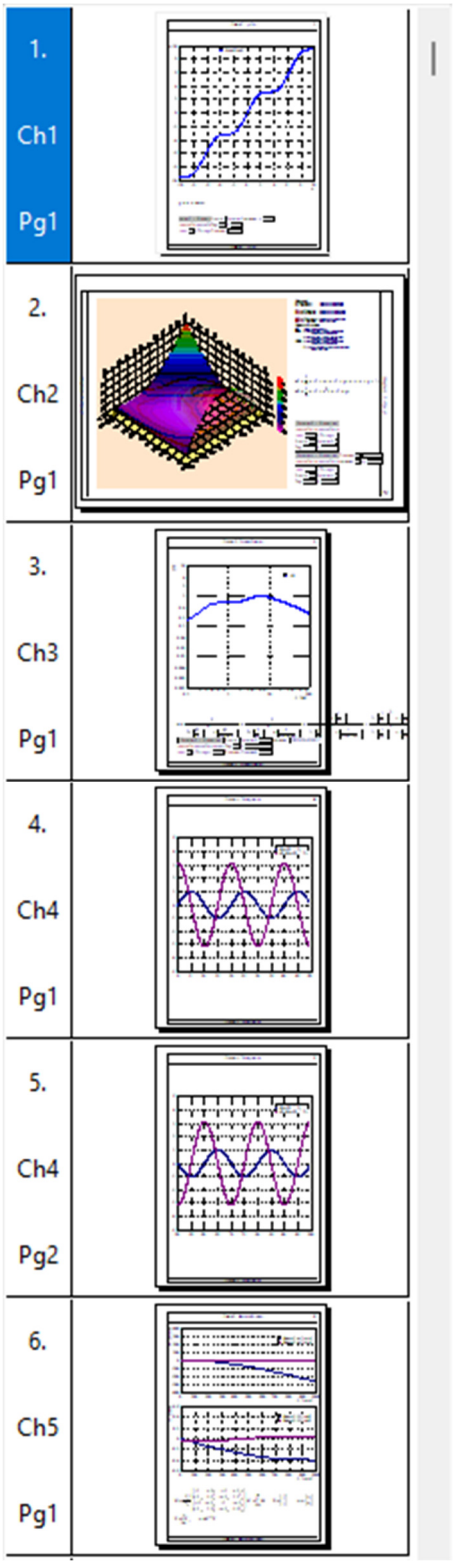
It is possible to draw via the `OnOutPaint`: `TRptComponentPaintEvent = procedure(Sender: TObject; Canvas: TCanvas; var PaintInfo: TPaintStruct) of Object` event into the component area.

There is also a client rectangle of the TLRptPaintBox component in the PaintInfo parameter.

Another drawing option is the assignment of the **Control**: TInfoCustomControl property. Then the area Control is drawn.

Another option is the assignment of the **Metafile**: TFileName property. The file content is drawn into the component client rectangular. The file content is not saved into the *.dfm file, only the file name and its path is saved. Out of these reasons, it is necessary to attach the file to the created application then.

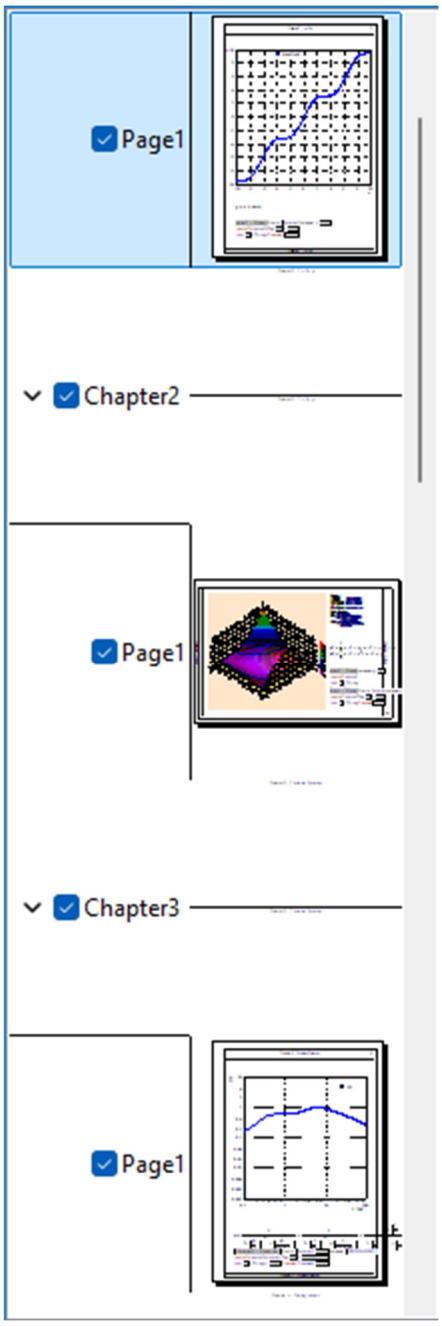
2.15 TPagesGrid = class(TCustomDrawGrid)



The TPagesGrid component is determined for the page view and page selection of the TLRpt component.

The component has the added property **LRpt**: TLRpt.

2.16 TPagesTreeView = class(TCustomTreeView)



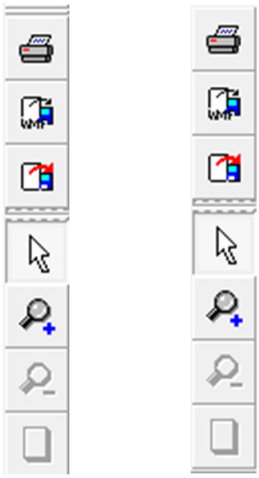
The TPagesTreeView component is determined for the chapters and page view with the option to select chapters and pages of the TLRpt component.

The component has the added property **LRpt**: TLRpt.

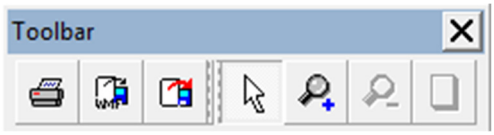
2.17 TLDock, TLToolbar, TLToolWindow, TLToolbarBtn, TLToolbarSep

It is possible to add the TLDock component on the window or frame area depending on its **Position** = (dpLeft, dpTop, dpRight, dpBottom) property.

The TLToolbar and TLWindow components may be docked according to the setting of the **DockedTo** property.



If the **DockedTo** property is not assigned in the TLToolbar a TLWindow components, then these components are depicted as a separate window.



It is possible to place objects of TLToolbarBtn and TLToolbarSep types on the TLToolbar component.

It is possible to determine the bit map of the TLToolbarBtn component via the **Images** and **ImageIndex** properties or the **Glyph** property.