TChart Component Properties (by Group)

Axis control:

AxisVisible	Boolean	Show / Hide the chart axis.
BottomAxis	TChartAxis	The horizontal bottom axis.
DepthAxis	TChartAxis	The depth (Z) axis.
LeftAxis	TChartAxis	The vertical left axis.
RightAxis	TChartAxis	The vertical right axis.
TopAxis	TChartAxis	The horizontal top axis.

3d Related:

Chart3dPercent	Integer	3d effect percent proportion.
View3d	Boolean	Show or hide the 3d effect.
View3dWalls	Boolean	Show or hide the 3d walls.

General properties:

AllowPanning	TPanningMode	Controls Right mouse button scrolling.
AllowZoom	Boolean	True to left mouse button zooming.
BackColor	TColor	The color used to fill the rectangle containing the Chart.
BackImage	TPicture	Assign an image as background to the chart.
BackImageInside	Boolean	Restricts an image to display inside the chart frame.
BackImageMode	TTeeBackImageMode	Decides how the chart background will display.
BufferedDisplay	Boolean	True means no flickering when drawing.
CancelMouse	Boolean	Toggles mouseclick sensitivity for zoom.
ClipPoints	Boolean	Shows/Hides points outside bounds.
TChartPen	The chart frame.	
Monochrome	Boolean	Draw the chart in monochrome (black and white) mode.
MonochromePrint	Boolean	Print the chart in monochrome mode.
OriginalCursor	TCursor	Returns original chart cursor.

Chart Legend:

Legend	TChartLegend	The Legend subcomponent.

Margins:

MarginBottom	Integer	The bottom chart margin percent.
MarginLeft	Integer	The left chart margin percent.
MarginRight	Integer	The right chart margin percent.
MarginTop	Integer	The top chart margin percent.

Series List:

SeriesList	TChartSeriesList	The chart Series.

Chart Title and Foot:

Title TChartTitle The top chart banner.

Foot TChartTitle The bottom chart banner.

Automatic Chart Paging:

MaxPointsPerPage Longint Maximum number of points per Page.

Page Longint The current Chart Page number.

NumPages Longint The total number of Pages.

See also:

TChart Component Reference

TChart public Methods

TDBChart Component Reference

List of Series Components

Canvas Property

Applies to
TChart, TDBChart components

Declaration
```
property Canvas : TCanvas3D;
```

Description
Use the Canvas property to access all Canvas and Canvas3D properties and methods.

Canvas example
```
//Draws a Line between co-ordinates 50,50 (
//from Chart Panel Top, Left) to 100,100
Chart1.Canvas.MoveTo(50,50);
Chart1.Canvas.LineTo(100,100);
```

More About TeeChart...

TeeChart Pro

TeeChart charting components have been written in Delphi by David Berneda. They are 100% Delphi Native VCL compliant. TeeChart products are distributed and supported by teeMach SL. of Barcelona, Catalonia, Spain.

Please email the following address for questions:

email address: info@teemach.com

www: http://www.teechart.com

Address:
teeMach SL.

Rocafort 35-37 5o3a

08015 BARCELONA

CATALONIA, (Spain)

Tel. +34 972 59 71 61

Fax: +34 972 59 71 75

Active Example

```
PointSeries1.Active:=not PointSeries1.Active;
```

Active Property

Example

Applies to

TChartSeries component

Declaration

```
property Active : Boolean;
```

Description

The Active property shows or hides the TChartSeries. It can be changed both at design time or runtime. When hiding, all point values are preserved, so there's no need to refill the values again when showing. The Series relatives chart axis are rescaled in order to accommodate changes.

ActiveSeriesLegend Method

Applies to
TChart, TDBChart components

Declaration
```
function ActiveSeriesLegend(SeriesIndex : Longint) : TChartSeries;
```

Description
The ActiveSeriesLegend function returns the nth Active Series.

If all Series are Active, calling then the above function is the same as using the Chart1.Series property.

Add Example

```
With Series1 do
Begin
  Add( 40, 'Pencil' , clRed ) ;
  Add( 60, 'Paper', clBlue ) ;
  Add( 30, 'Ribbon', clGreen ) ;
end;
```

Add Method

See also Example

Applies to

TChartSeries component

Declaration

```
function Add(Const AValue:Double; Const ALabel:String;
AColor:TColor):Longint; virtual;
```

Description

The TChartSeries Add method can be used to insert new Series points when you do not have an X Value for the point.

This function inserts a new point in the Series. The new point only has Y values. X value is automatically calculated.

The AXLabel parameter is optional (can be empty '').

The AColor parameter is optional (can be clTeeColor).

The function returns the new point position in the Values list.

See Also
TChartSeries.AddY

AddNull Method

See also

Applies to

TChartSeries component

Declaration

```
function AddNull(Const ALabel:String):Longint; virtual;
```

Description

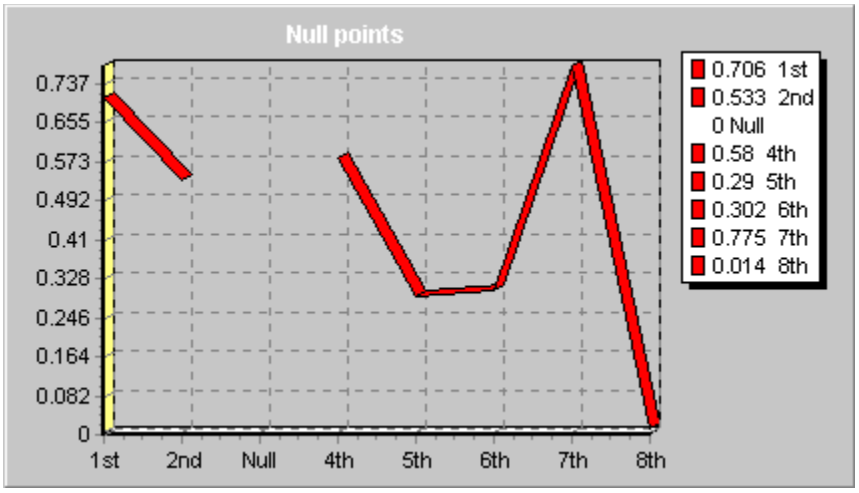
The TChartSeries AddNull method can be used to insert new Series **Null** points when you do not have an X Value for the point.

This function inserts a new empty point in the Series. Series points before and after do not connect across the space left by the Null point.

The function returns the new point position in the Values list.

Example

In this example the Axis and Legend labels ("1st", "2nd", "null", etc) are generated by the ALabel parameter of the Add and AddNull methods to clarify how points are added. In your own Chart, creating and displaying Labels is optional.



See Also

TChartSeries.Add

TChartSeries.AddY

AddArrow Example

This code fills an ArrowSeries with two random arrows:

```
var
  x0,y0,x1,y1 : Double;
begin
  ArrowSeries1.StartXValues.DateTime:=False;
  ArrowSeries1.EndXValues.DateTime:=False;
  ArrowSeries1.Clear;
  x0 :=Random( 1000 );
  y0 :=Random( 1000 );
  x1 :=x0 + Random( 1000 ) - 500 ;
  y1 :=y0 + Random( 1000 ) - 500 ;
  ArrowSeries1.AddArrow( x0, y0, x1, y1, '', clBlue);
  x0 :=Random( 1000 );
  y0 :=Random( 1000 );
  x1 :=x0 + Random( 1000 ) - 500 ;
  y1 :=y0 + Random( 1000 ) - 500 ;
  ArrowSeries1.AddArrow( x0, y0, x1, y1, '', clYellow);
end;
```

AddArrow Method

See also Example

Applies to

TArrowSeries component

Declaration

```
function AddArrow(Const X0, Y0, X1, Y1: Double; Const ALabel: String;
AColor: TColor): Longint;
```

Description

The AddArrow function adds a new arrow point to the Series. Returns the position of the Arrow in the list. Positions start at zero.

Each arrow is made of 2 points:

(X0,Y0) The starting arrow point.

(X1,Y1) The arrow head end point.

See Also

TArrowSeries.StartXValues

TArrowSeries.StartYValues

TArrowSeries.EndXValues

TArrowSeries.EndYValues

AddBar Method

See also

Applies to

TBarSeries component

Declaration

```
function AddBar(Const Value: Double; Const Text: String; Color: TColor) :
Longint;
```

Description

***Important** The AddBar Method has been maintained only for this version of TeeChart to help those migrating from previous versions of TeeChart. The AddBar method calls the Add method. For those of you writing new code please use the Add method. The AddBar method will be removed in the next version of TeeChart.

The AddBar method appends a new Bar to the Series Points List. The Bar point is assigned the Value, Text and Color parameters. You can specify clTeeColor as Color parameter to draw it using a predefined color. The Text parameter is used to draw Axis Labels, Bar Marks and Legend. If you want to specify exact X coordinates, then you must use TChartSeries.AddXY method.

AddBar Method

See also

Applies to

THorizBarSeries component

Declaration

```
function AddBar( Const Value: Double; Const Text: String; Color: TColor): LongInt;
```

Description

***Important** The AddBar Method has been maintained only for this version of TeeChart to help those migrating from previous versions of TeeChart. The AddBar method calls the Add method. For those of you writing new code please use the Add method. The AddBar method will be removed in the next version of TeeChart.

The AddBar function appends a new horizontal Bar point to the Series.

The Value parameter defines the Bar width.

New added points are optionally sorted in ascending or descending order either by their horizontal or vertical coordinates.

You can turn sorting off by setting the XValues and YValues Order property to loNone.

This function returns the corresponding internal Bar point index. You can then refer to this Bar point by using this index.

It calls the default AddXY method.

See Also

TChartSeries.AddY

TChartSeries.Add

TChartSeries.XValue

TChartSeries.YValue

TChartSeries.XLabel

TChartSeries.ValueColor

See Also

TChartSeries.AddXY

TChartValueList.Order

Example

This example randomly creates bubble values.

```
procedure TBubbleForm.FormCreate(Sender: TObject);
var t:Longint;
begin
 ComboBox1.ItemIndex:=Ord(psCircle); { <-- Circled Bubbles by default }
 BubbleSeries1.Clear;
 for t:=1 to 100 do
    BubbleSeries1.AddBubble( t,
                    Random(ChartSamplesMax),        { <-- y value }
                    ChartSamplesMax/(20+Random(25)), { <-- radius value }
                    '',                     { <-- label string }
                    GetDefaultColor(t));          { <-- color }
end;
```

AddBubble Method

See also Example

Applies to

TBubbleSeries component

Declaration

```
function AddBubble(Const AX,AY,ARadius:Double; Const AXLabel:String;
AColor:TColor): Longint;
```

Description

The AddBubble method appends a new Bubble point to the Series Points List. The Bubble point is assigned to be at AX,AY coordinates and have ARadius, Label and Color parameters. You can specify clTeeColor as Color parameter to draw it using a predefined color. The Label parameter is used to draw Axis Labels, Bubble Marks and Legend.

See Also

TChartSeries.AddXY

TBubbleSeries.RadiusValues

TChartSeries.XLabel

TChartSeries.XValue

TChartSeries.YValue

TChartSeries.ValueColor

AddGantt Method

See also Example

Applies to

TGanttSeries component

Declaration

```
function AddGantt(Const AStart,AEnd,AY: Double; Const AXLabel: String): LongInt;
```

Description

The AddGantt function adds a new Gantt bar to the Series. Each Gantt bar has the following parameters: AStart and AEnd (the starting and ending Gantt bar horizontal coordinates). These can be normal floating values or DateTime values.

AY is the vertical Gantt bar coordinate. You can add as many Gantt bars you want, with or without same vertical coordinate. The vertical coordinate is usually a value starting from zero, but you can choose any other arbitrary value.

AXLabel is the associated Gantt bar text. It will be optionally shown as a Label at vertical axis and at TGanttSeries.Marks.

You can assign connecting lines between Gantt bars by using the TGanttSeries.NextTask property. You can assign a specific color to each Gantt bar by using the TChartSeries.ValueColor property or by calling the TGanttBar.AddGanttColor method. This function returns the corresponding internal point index. You can then refer to this Gantt bar by using this index. New added points are optionally sorted in ascending or descending order either by their horizontal or vertical coordinates. You can turn sorting off by setting the XValues and YValues Order property to loNone.

AddGantt method Example

Example

This code adds several Gantt bars and connects them:

```
With GanttSeries1 do
begin
{ remove all points }
  Clear;
{ add several bars }
  AddGantt( EncodeDate(1997,1,1), EncodeDate(1997,1,31),2,'Hello');
  AddGantt( EncodeDate(1997,1,15), EncodeDate(1997,2,15),1,'Nice');
  AddGantt( EncodeDate(1997,2,1), EncodeDate(1997,2,28),0,'World');
  AddGantt( EncodeDate(1997,3,1), EncodeDate(1997,3,31),2,'');
  AddGantt( EncodeDate(1997,4,1), EncodeDate(1997,4,30),0,'');
  AddGantt( EncodeDate(1997,3,15), EncodeDate(1997,4,15),1,'');
{ connect some bars }
  NextTask[0]:=3;
  NextTask[1]:=5;
{ change connecting lines }
  ConnectingPen.Width:=2;
  ConnectingPen.Color:=clBlue;
{ increase bar heights }
  Pointer.VertSize:=16;
{ remove legend }
  ParentChart.Legend.Visible:=False;
{ arrange bottom axis to show exact datetime labels }
  GetHorizAxis.ExactDateTime:=True;
  GetHorizAxis.Increment:=DateTimeStep[dtOneMonth];
end;
```

See Also

TGanttSeries.NextTask

TChartSeries.ValueColor

TGanttBar.AddGanttColor

TChartValueList.Order

AddGanttColor Method

See also

Applies to

TGanttSeries component

Declaration

```
function AddGanttColor(Const AStart, AEnd, AY: Double; Const AXLabel:
String; AColor: TColor): LongInt;
```

Description

The AddGanttColor function adds a new Gantt bar point to the Series.

It calls the AddGantt function and sets the specified AColor parameter by using the TChartSeries.ValueColor property.

This function returns the corresponding internal point index. You can then refer to this Gantt bar by using this index.

Please refer to TGanttSeries.AddGantt function for an example of adding Gantt bars.

See Also

TGanttSeries.AddGantt

TGanttSeries.NextTask

TChartSeries.ValueColor

AddPie Example

This example adds 3 pie sectors to series1 (a pie series)

```
With Series1 do
Begin
  Clear ;
  AddPie( 40, 'Pencil' , clRed ) ;
  AddPie( 60, 'Paper', clBlue ) ;
  AddPie( 30, 'Ribbon', clGreen ) ;
end;
```

AddPie Method

Example

Applies to

TPieSeries component

Declaration

```
function AddPie(Const PieValue: Double; Const APieLabel: String; AColor:
TColor) : LongInt;
```

Description

***Important** The AddPie Method has been maintained for this version only of TeeChart to help those migrating code from previous versions of TeeChart. The AddPie method now calls the Add method. For those of you writing new code please use the Add method directly. The AddPie method will be removed in the next version of TeeChart.

AddPie appends new Pie sectors to a TPieSeries component.

The PieValue can be any integer.

AddSeries Method

See also

Applies to

TChart component

Declaration

```
procedure AddSeries(ASeries : TChartSeries);
```

Description

The AddSeries method adds a new Series component to the Chart. It's exactly the same as setting the Series.ParentChart property to the Chart.

```
LineSeries1.ParentChart := Chart1;
```

You should use the ParentChart property if you add the Series at runtime.

See Also
TChartSeries.ParentChart

AddValue Method

Applies to
TChartSeries component

Declaration
```
procedure AddValue( ValueIndex : Longint ) ; virtual ;
```

Description
You don't need to call AddValue directly.

This is an internal method.

AddXY Example

```
procedure TCurveFittingForm.Timer1Timer(Sender: TObject);
begin
 Timer1.Enabled:=False; { <-- stop timer }
 With StockPrice do
 Begin
   Delete(0); { <-- remove the first point }
   { Add a new random point }
   AddXY( XValues.Last+1,
         (YValues.Last/YValues.Multiplier)+(Random(ChartSamplesMax)-(ChartSamplesMax/2)),
         '',clTeeColor);
   Chart1Zoom(Self); { <-- recalculate Curve !!!! }
 end;
 Timer1.Enabled:=True; { <-- restart timer }
end;
```

AddXY Method

See also Example

Applies to

TChartSeries component

Declaration

```
function AddXY(Const AXValue, AYValue: Double; Const AXLabel: String;
AColor: TColor) : Longint;
```

Description

This function inserts a new point in the Series. The new point has X and Y values. The AXLabel parameter is optional (can be empty ''). The AColor parameter is optional (can be clTeeColor). The function returns the new point position in the Values list.

See Also
AddY function

AddY Example

```
Series1.AddY( 6.5, 'Revenues', clTeeColor);
```

AddY Method

See also Example

Applies to

TChartSeries component

Declaration

```
function AddY(Const AYValue: Double; Const AXLabel: String; AColor: TColor): Longint;
```

Description

The TChartSeries AddY method can be used to insert new Series points when you do not have an X Value for the point.

This function inserts a new point in the Series. The new point only has Y values. X value is automatically calculated.

The AXLabel parameter is optional (can be empty '').

The AColor parameter is optional (can be clTeeColor).

The function returns the new point position in the Values list.

See Also

AddXY Method

TChartSeries.Add

AdjustFrame Property

See also

Applies to

TChartTitle component

Declaration

```
property AdjustFrame : Boolean;
```

Description

Default: True.

The AdjustFrame property controls if Chart Title and Foot frames will be resized to full Chart dimensions or to the title text width. It has only effect when Chart.Title or Chart.Foot.Frame.Visible property is True.

See Also

TChart.Title

TChart.Foot

TChartTitle.Frame

AdjustMaxMin Method

See also

Applies to

TChartAxis component

Declaration

```
procedure AdjustMaxMin;
```

Description

This Axis method calculates Maximum and Minimum values based on Max and Min values of the dependent Series.

AdjustMaxMin is automatically called if Axis.Automatic is True.

The Chart UndoZoom method calls AdjustMaxMin for Left, Right,Top and Bottom axis.

See Also
Chart UndoZoom

AfterDrawValues Event

Applies to
TChartSeries component

Declaration
```
property AfterDrawValues : TNotifyEvent;
```

Description
AfterDrawValues event is called just after the Series points have been painted. You can then paint on the Chart.Canvas your customized drawings.

WARNING:
Do not change Series properties that would force the Series to be repainted. This may cause recursion and endless repainting !

Alignment Property

See also

Applies to

TChartLegend component

Declaration

```
property Alignment : TLegendAlignment;
```

Description

Default Value: laRight

The Alignment property defines the Legend position.

Legend can be currently placed at Top, Left, Right and Bottom side of Chart.

Left and Right Legend alignments define a vertical Legend with currently one single column of items.

Top and Bottom Legend alignments define an horizontal Legend with currently one single row of items.

The Legend itself automatically reduces the number of displayed legend items based on the available charting space.

The Legend.ResizeChart property controls if Legend dimensions should be used to reduce Chart points space.

The Legend.OnGetLegendRect event provides a mechanism to supply the desired Rectangle Legend dimensions and placement.

The Legend.OnGetLegendPos event can be used to specify fixed Legend items X Y coordinates.

The Legend.HorizMargin and VertMargin properties control distance between Legend and Left and Right margins.

The Legend.TopPos property can be used in Left or Right Legend alignments to control vertical distance between Legend and Top Chart Margin.

These techniques allow almost complete Legend control.

TChartLegend is quite a big component with many available formatting and dimensioning properties.

Alignment Property (TChartTitle)

See also

Applies to

TChartTitle component

Declaration

```
property Alignment : TAlignment;
```

Description

Default Value: taCenter

The Alignment property controls how Chart Title and Foot text will be aligned within Chart rectangle.

A TChartTitle can optionally be surrounded by a Frame.

See Also

TChartLegend

TChartLegend.HorizMargin

TChartLegend.ResizeChart

TChartLegend.TopPos

TChartLegend.VertMargin

See Also
TChartTitle.Frame

AllowPanning Property

See also

Applies to

TChart, TDBChart components

Declaration

```
property AllowPanning : TPanningMode
```

Description

The AllowPanning property controls runtime Scrolling.

Possible values are:

pmNone	Deny scrolling.
pmHorizontal	Allow only Horizontal Scrolling.
pmVertical	Allow only Vertical Scrolling.
pmBoth	Allow complete Horizontal and Vertical Scrolling.

See Also
Scrolling and Panning

AllowSinglePoint Property

See also

Applies to
TChartSeries component

Declaration
```
property AllowSinglePoint : Boolean;
```

Description
(Advanced)
The AllowSinglePoint public property controls if a given Series type can display a single point or needs more than one point to draw.
TLineSeries, TAreaSeries and TFastLineSeries components set AllowSinglePoint property to False, as they need at least two points to draw.
This property can be of interest to Series components developers.
.

See Also

TChartSeries.DrawBetweenPoints

TCustomSeries.ClickableLine

TChartSeries.CalcVisiblePoints

AllowZoom Example

You can also Zoom a specific chart region by coding:

(This example uses the Bottom and Left axis)

```
Chart1.BottomAxis.Automatic := False ;
Chart1.BottomAxis.Minimum  := 45.2 ;
Chart1.BottomAxis.Maximum  := 67.1 ;
Chart1.LeftAxis.Automatic := False ;
Chart1.LeftAxis.Minimum  := 150 ;
Chart1.LeftAxis.Maximum  := 300 ;
```

AllowZoom Property

See also Example

Applies to

TChart, TDBChart components

Declaration

```
property AllowZoom : Boolean;
```

Description

The AllowZoom boolean property toggles the runtime Zoom. Setting it to True will allow runtime Zoom by dragging the mouse.

See Also
Zooming

Angle Property

See also

Applies to
TChartAxisTitle component

Declaration

```
property Angle : Integer;
```

Description

The Angle property determines the Axis Title rotation in degree units.

Valid values are 0, 90, 180, 270 and 360 degrees.

TChartAxis.Title.Caption property should be non empty.

By default, horizontal Axis (TopAxis and BottomAxis) have a Title Angle of 0.

LeftAxis.Title.Angle is 90 by default.

RightAxis.Title.Angle is 270 by default.

Warning:
Rotated Fonts are partially supported in Windows 16bit GDI Metafile format. Some printer devices can't print rotated fonts. Check you have the latest Printer Driver version.

See Also

TChartAxis.LabelsAngle

TChartAxisTitle.Caption

AngleToPos Method

See also

Applies to

TCircledSeries component

Declaration

```
function AngleToPos(Const Angle; AXRadius; AYRadius: Double;var X,Y: Longint);
```

Description

The AngleToPos functions returns the exact Screen position for a given pair of Angle and Radius values.

Angles must be expressed in radians from 0 to 2*PI. Radius parameter is in number of pixels.

See Also

TCircledSeries.PointToAngle

TCircledSeries.RotationAngle

TCircledSeries.XRadius

TCircledSeries.YRadius

AnimatedZoom Property

See also

Applies to
TChart, TDBChart component

Declaration
```
property AnimatedZoom : Boolean;
```

Description
Default False

The AnimatedZoom property determines if Zoom will be performed directly or it will be displayed as an animated sequence of zooms. The AnimatedZoomSteps property controls the number of zoom steps. AnimatedZoom can be useful when there are a lot of points.

This boolean property controls if Zoom will be instantaneous or it will be dynamic.

When AnimatedZoom is True, Zoom is performed by applying successive Zooms. The number of Zoom steps can be changed using the AnimatedZoomSteps property.

See Also

TChart.AllowZoom

TChart.AnimatedZoomSteps

TChart.ZoomPercent

TChart.ZoomRect

AnimatedZoomFactor Global Variable

See also

Applies to
All TeeChart components

Declaration
```
property AnimatedZoomFactor : Double;
```

Description
Default Value: 3.0

The AnimatedZoomFactor property defines the zoom proportion ratio to apply at each successive zoom step.

The TChart.AnimatedZoom property must be True for this property to be in effect.

The TChart.AnimatedZoomSteps property determine the number of zoom steps to apply until Axis scales are set to the desired zoom.

Change the AnimatedZoomFactor property to make those steps to go faster or slower.

See Also

TChart.AnimatedZoom

TChart.AnimatedZoomSteps

AnimatedZoomSteps Example

This code turns on the animated zoom feature:

```
Chart1.AnimatedZoom := True ;
Chart1.AnimatedZoomSteps := 4 ;
```

AnimatedZoomSteps Property

See also Example

Applies to

TChart, TDBChart components

Declaration

```
property AnimatedZoomSteps : Integer;
```

Description

Default: 8

The AnimatedZoomSteps property determines the number of steps of the animated zooming sequence. Big number of steps can delay zooming. The AnimatedZoom property should be True.

See Also

TChart.AllowZoom

TChart.AnimatedZoom

TChart.ZoomPercent

TChart.ZoomRect

ApplyBarOffset Method

See also

Applies to

TCustomBarSeries component

Declaration

```
function ApplyBarOffset(Position : Longint): LongInt;
```

Description

The ApplyBarOffset function modifies and returns the Position parameter. BarSeries have an OffsetPercent property that makes all Bars to be displayed horizontally a specific amount of pixels based on Bar's width.

The Position parameter must be expressed in screen pixels.

The formula used is:

```
result:= Position + Round( OffsetPercent * BarWidth / 100.0 )
```

When OffsetPercent property is zero (the default), the Position parameter is returned unmodified.

See Also

TCustomBarSeries.OffsetPercent

TCustomBarSeries.BarWidth

ApplyDark Method

See also

Applies to

All TeeChart Components

Declaration

```
function ApplyDark(Color: TColor; HowMuch: Byte) : TColor;
```

Description

Global

The ApplyDark function converts the Color parameter to a darker color. The HowMuch parameter indicates the quantity of dark increment. It is used by TBarSeries and TPieSeries to calculate the right color to draw Bar sides and Pie 3D zones. Series.pas must be in your uses clause.

This is the ApplyDark source code:

```
Function ApplyDark(Color:TColor; HowMuch:Byte):TColor;
Var r,g,b:Byte;
Begin
  Color:=ColorToRGB(Color);
  r:=GetRValue(Color);
  g:=GetGValue(Color);
  b:=GetBValue(Color);
  if r>HowMuch then r:=r-HowMuch else r:=0;
  if g>HowMuch then g:=g-HowMuch else g:=0;
  if b>HowMuch then b:=b-HowMuch else b:=0;
  result:=RGB(r,g,b);
End;
```

See Also

TCustomBarSeries.Dark3D

TPieSeries.Dark3D

ApplyZOrder Property

See also

Applies to

TChart component

Declaration

```
property ApplyZOrder : Boolean;
```

Description

Run-time only.

The ApplyZOrder property controls if several Series on the same TChart component should be displayed each one in a different Z space.

It's valid only when TChart.View3D property is True and when there's more than one Series in same chart.

When False, all Series are drawn using the full Chart Z space. The Chart output can be confusing if Series overlap.

The "LASTVALU.PAS" example unit under TEEDEMO.DPR project shows an example of ApplyZOrder property.

See Also

TChart.MaxZOrder

TChart.View3D

TChartSeries.ZOrder

AreaBrush Property

See also

Applies to

TAreaSeries component

Declaration

```
property AreaBrush : TBrushStyle;
```

Description

Default Value: bsSolid

The AreaBrush property indicates the kind of Brush used to fill the background Area region. You can control the Area background color by using the AreaColor property.

See Also

TAreaSeries.SeriesColor

TAreaSeries.AreaColor

TAreaSeries.AreaLinesPen

Color Property (TChartTitle)

Applies to
TChartTitle component

Declaration
```
property Color : TColor;
```

Description
Default Value: clTeeColor

Color used to fill the Chart Title background..

AreaColor Property

See also

Applies to

TAreaSeries component

Declaration

```
property AreaColor : TColor;
```

Description

Default Value: clTeeColor

The AreaColor property defines the Color used to fill the background Area region. You can control the Brush style by using the AreaBrush property.

See Also

TAreaSeries.SeriesColor

TAreaSeries.AreaBrush

TAreaSeries.AreaLinesPen

AreaLinesPen Property

See also

Applies to

TAreaSeries component

Declaration

```
property AreaLinesPen : TChartPen;
```

Description

The AreaLinesPen property indicates the kind of pen used to draw vertical lines across the Area region. By default AreaLinesPen.Visible is False, so you need first to set it to True. You can control the Area Brush style by using the AreaBrush property.

See Also

TAreaSeries.SeriesColor

TAreaSeries.AreaBrush

TAreaSeries.AreaColor

Arrow Property

See also

Applies to

TSeriesMarks component

Declaration

```
property Arrow : TChartPen;
```

Description

The Arrow property determines the kind of pen used to draw a line connecting the Point Mark to the corresponding Series point.

Each Series component handles Marks in a different manner, thus the Arrow coordinates are specific to each Series type.

By default, Arrow pen is defined to be a White solid pen of 1 pixel width.

See Also
TSeriesMarks.ArrowLength

ArrowCha Unit

The ArrowCha unit contains the declarations for the following components and for the enumerated type associated with them. When you add a component declared in this unit to a form, the unit is automatically added to the uses clause of that form's unit. The following items are declared in the this unit:

Components

TArrowSeries

Types

Routines

To see a listing of items declared in this unit including their declarations, use the Project Browser.

ArrowHeight property example

This code checks series type and changes the ArrowHeight accordingly. Here acting like on/off.

```
if TheSeries is TCustomBarSeries then
   TheSeries.Marks.ArrowHeight:=10
 else
   TheSeries.Marks.ArrowHeight:=0;
```

ArrowHeight Property

See also Example

Applies to

TArrowSeries component

Declaration

```
property ArrowHeight : Integer;
```

Description

Default value: 4

The ArrowHeight property determines the vertical arrow head size in pixels.

See Also

TArrowSeries.ArrowWidth

TSeriesPointer.VertSize

ArrowLength property example

This code checks series type and changes the ArrowLength.

```
if TheSeries is TCustomBarSeries then
   TheSeries.Marks.ArrowLength:=10
 else
   TheSeries.Marks.ArrowLength:=0;
```

ArrowLength Property

See also Example

Applies to

TSeriesMarks component

Declaration

```
property ArrowLength : Integer;
```

Description

Default Value: 16

The ArrowLength property determines the number of pixels used to display a line connecting the Series Marks to their corresponding points.

The Arrow property is the Pen used to draw this line.

See Also
TSeriesMarks.Arrow

ArrowWidth Property

See also

Applies to

TArrowSeries component

Declaration

```
property ArrowWidth : Integer;
```

Description

Default value: 4

The ArrowHeight property determines the horizontal arrow head size in pixels.

See Also

TArrowSeries.ArrowHeight

TArrowSeries.ArrowLength

TSeriesPointer.HorizSize

Assign method example

This code uses a button to copy a series from one chart to another.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
CopySeries(DBChart2,DBChart1, self);
BitBtn1.Visible:=False;
end;
type TChartClass=class of TChart;
Procedure TForm1.CopySeries(DestChart,SourceChart:TChart;
  AOwner:TComponent);
Var tmpSeries:TChartSeries;
  tmpS:TChartSeriesClass;
  t:Longint;
begin
 for t:=0 to SourceChart.SeriesCount-1 do
 begin
  tmpS:=TChartSeriesClass(SourceChart.Series[t].ClassType);
  tmpSeries:=tmpS.Create(AOwner);
  tmpSeries.Assign(SourceChart.Series[t]);
  tmpSeries.Name:=(SourceChart.Series[t].Name) + 'copy';
  DestChart.AddSeries(tmpSeries);
 end;
end;
```

Assign Method

See also Example

Applies to

TChart, TDBChart components

Declaration

```
Procedure Assign(Source : TPersistent) : Override
```

Description

The Assign method copies all properties from a Series component to another.

Only the common properties shared by both source and destination Series are copied.

The following code copies all properties from Series2 into Series1:

Series1.Assign(Series2) ;

Some Series types restore property values after assigning them. For example, TPointSeries restores the Pointer.Visible property to True after being assigned to a TLineSeries, which has Pointers invisible by default.

Note: Series events are not assigned. Series DataSource and FunctionType properties are assigned. Assign is used by CloneChartSeries and ChangeSeriesType methods for example.

See Also

AssignSeries

TChartSeries.AssignValues

AssignSeries Global Method

See also

Unit
Chart

Applies to
All TeeChart Components

Declaration

```
Procedure AssignSeries(Var OldSeries,NewSeries:TChartSeries);
```

Description
Global

The AssignSeries method copies all properties and event handlers from OldSeries to NewSeries, then removes OldSeries and finally adds NewSeries to the Chart.

Data point values are not assigned, but as DataSource property is copied, NewSeries obtains points from DataSource.

This method is used internally when changing a Series type to assign old properties to new Series type instance.

See Also

TChart.Assign

TChartSeries.AssignValues

AssignValues Example

This code copies all points from LineSeries1 to LineSeries2 :

```
LineSeries2.AssignValues( LineSeries1 );
```

AssignValues Method

See also Example

Applies to

TChartSeries component

Declaration

```
procedure AssignValues(Source : TChartSeries);
```

Description

The AssignValues method creates new Series points from Source list of points.

Source:

```
Procedure TChartSeries.AssignValues(Source : TChartSeries);
var t:Longint;
Begin
 Clear;
 for t:=0 to Source.Count-1 do AddedValue(Source,t);
end;
```

The AddedValue method creates a new clone point from Source point and inserts it using the AddXY method.

See Also

TChartSeries.AddXY

TChartSeries.RecalcOptions

TChartSeries.RefreshSeries

TChart.Assign

AutoMarkPosition Property

See also

Applies to

TBarSeries and THorizBarSeries components

Declaration

```
property AutoMarkPosition : Boolean;
```

Description

Default: True

The AutoMarkPosition property controls if Marks on TBarSeries will be repositioned if there's the possibility of Mark overlapping. Marks are displaced to the top of the Bars to minimize the overlapping effect of Marks with long text or big font sizes.

When False, no checking is performed and all Marks are drawn at their Mark.ArrowLength distance to the top of the Bar.

See Also

TBarSeries.BarWidth

THorizBarSeries.BarHeight

AutoRefresh Property

See also

Applies to
TDBChart component

Declaration
```
property AutoRefresh : Boolean;
```

Description
Default Value: True

The AutoRefresh property determines if TDBChart will load all Table or Query records in an automatic way just when the Table or Query are Opened.

When False, you should manually force record retrieval by calling one of the following methods:

```
TDBChart.RefreshDataSet
TDBChart.RefreshData
```

When True, TDBChart will retrieve all Series point values as soon the Table is opened or the Query is executed.

See Also

TDBChart.RefreshData

TDBChart.RefreshDataSet

Automatic Example

You can control the Axis scale manually:

```
Chart1.LeftAxis.Automatic := False ;
Chart1.LeftAxis.Maximum  := 1000 ;
Chart1.LeftAxis.Minimum  :=  0 ;
```

Or, in case DateTime values:

```
Chart1.BottomAxis.Automatic := False ;
Chart1.BottomAxis.Maximum  := EncodeDate ( 1995, 12 , 31 ) ;
Chart1.BottomAxis.Minimum  := EncodeDate ( 1995, 1 , 1 ) ;
```

Automatic Property

Example

Applies to

TChartAxis component

Declaration

```
property Automatic : Boolean;
```

Description

If Axis.Automatic property is True, then the Maximum and Minimum values for the Axis will be calculated with all Axis dependent Series.

AutomaticMaximum Property

See also

Applies to
TChartAxis component

Declaration
```
property AutomaticMaximum : Boolean
```

Description
Default: True

The AutomaticMaximum property controls if Axis will adjust the Maximum value automatically based on the maximum value of its associated Series.

Example
```
With Chart1.LeftAxis do
begin
 Automatic := False;
 AutomaticMaximum := True;
 Minimum := 500;
end;
```

See Also

TChartAxis.Automatic

TChartAxis.AutomaticMinimum

TChartAxis.Maximum

TChartAxis.Minimum

AutomaticMinimum Property

See also

Applies to

TChartAxis component

Declaration

```
property AutomaticMinimum : Boolean;
```

Description

Default True

The AutomaticMinimum property controls if Axis will adjust the Minimum value automatically based on the minimum value of its associated Series.

Example

```
With Chart1.LeftAxis do
begin
 Automatic := False;
 Maximum := 1000;
 AutomaticMinimum := True;
end;
```

See Also

TChartAxis.Automatic

TChartAxis.AutomaticMaximum

TChartAxis.Maximum

TChartAxis.Minimum

Axis Property

See also

Applies to

TChartAxis component

Declaration

```
property Axis : TChartPen;
```

Description

The Axis property determines the kind of pen used to draw the Axis major lines. These are the lines which go from Axis Minimum to Axis Maximum screen positions.

See Also

TChartAxis.Grid

TChartAxis.GridCentered

TChartAxis.Ticks

TChartAxis.MinorTicks

TChartAxis.TicksInner

TChartAxis.LabelsFont

AxisValuesFormat property Example

AxisValuesFormat examples:

```
1.  DBChart1.LeftAxis.AxisValuesFormat:='#,##0.00'
    {Outputs labels eg. 10.00, 25.00, 150.00}
2.    DBChart1.LeftAxis.AxisValuesFormat:='#,##0.###'
    {Outputs labels eg. 10, 25, 150}
```

This code can be used at the OnGetAxisLabel Chart event to paint labels only at the desired increments. The following code is useful for logarithmic axis:

```
{event OnGetAxisLabel - see Object Inspector events for TChart/TDBChart}
procedure TForm1.DBChart1GetAxisLabel(Sender: TChartAxis;
 Series: TChartSeries; ValueIndex: Longint; var LabelText: string);
 Function MyLabelValue(Const Value:Double):String;
 begin
  result:=FormatFloat(Sender.AxisValuesFormat,Value);
 end;
begin
 if ( Sender = DBChart1.LeftAxis ) then
 With Sender do
 begin
  if ( LabelText<>MyLabelValue(10) ) and
    ( LabelText<>MyLabelValue(100) ) and
    ( LabelText<>MyLabelValue(1000) ) and
    ( LabelText<>MyLabelValue(10000) ) then
     LabelText:='';
 end;
end;
```

AxisValuesFormat Property

See also Example

Applies to

TChartAxis component

Declaration

```
property AxisValuesFormat : String;
```

Description

Default Value: '#,##0.###'

The AxisValuesFormat specifies the desired formatting string to be applied to Axis Labels. It has effect when Axis associated Series have their XValues.DateTime or YValues.DateTime property set to False.

For DateTime Axis labels use the TChartAxis.DateTimeFormat property.

See Also

TChartAxis.DateTimeFormat

TChartAxis.OnGetAxisLabel

TChartAxis.LabelStyle

TChartAxis.LabelsMultiline

TChartValueList.DateTime

TChartAxis.LabelsFont

LabelsMultiline Property (TChartAxis)

See also

Applies to
TChartAxis component

Declaration
```
property LabelsMultiline : Boolean;
```

Description
Default = False

Enables multiline Axis labels.

When True, spaces in point Labels or in Axis DateTimeFormat / ValueFormat are used to break the label in more than one line of text).

When False, labels with #13 characters are break in lines.

Example
```
Series1.Add( 1234, 'Hello'+#13+'World', clGreen );
```

Example for DateTime labels

The following will show the bottom axis labels in two lines of text, one

showing the month and day, and the second line showing the year:

Feb-28 Mar-1 ..

1998 1998 ..

```
Series1.AddXY( EncodeDate( 1998,2,28 ), 100, '', clTeeColor );
Series1.AddXY( EncodeDate( 1998,3,1 ), 200, '', clTeeColor );
Series1.AddXY( EncodeDate( 1998,3,2 ), 150, '', clTeeColor );
Series1.XValues.DateTime := True;
Chart1.BottomAxis.DateTimeFormat := 'mm/dd hh:mm';  { <-- space }
```

If you set the Axis.LabelsMultiLine property to True, then the axis will

automatically split labels in lines where it finds an space.

```
Chart1.BottomAxis.LabelsMultiLine:=True;
```

Will use the formatting divided in two:

'mm/dd' for the first line

'hh:mm' for the second line

At run-time you can always split the label into lines programmatically, using the OnGetAxisLabel event:

```
procedure TForm1.Chart1GetAxisLabel(Sender: TChartAxis;
 Series: TChartSeries; ValueIndex: Integer; var LabelText: String);
begin
 TeeSplitInLines(LabelText,' ');
end;
```

Multi-line DateTime axis labels:

The global "TeeSplitInLines" procedure converts all spaces in "LabelText" to

line separators (returns).

The axis LabelsAngle property (label rotation in degree angles 0, 90, 180 or 270), can

also be used with multi-line axis labels.

See Also

TChartAxis.DateTimeFormat

TChartAxis.OnGetAxisLabel

TChartAxis.LabelStyle

TChartAxis.AxisValuesFormat

TChartValueList.DateTime

TChartAxis.LabelsFont

AxisVisible Property

Applies to
TChart, TDBChart components

Declaration
```
property AxisVisible : Boolean;
```

Description
Default value: True

This property shows or hides the four Chart Axis at once.

Each Axis will be drawn depending also on their Visible property.

BackColor Property

See also

Applies to

TChart, TDBChart components

Declaration

```
property BackColor : TColor;
```

Description

The BackColor property is the color the Chart rectangle is filled with. The Chart rectangle is the screen area between axis. Setting BackColor to clTeeColor does not fill the above rectangle. Assigning clTeeColor to Chart.BackColor makes TeeChart to not fill the Chart back area, so Gradient fills and Background Bitmaps can be shown.

BackColor Property

See also

Applies to

TSeriesMarks component

Declaration

```
property BackColor : TColor;
```

Description

Default Value: ChartMarkColor = $80FFFF (Yellow)

The BackColor property defines the color used to fill the Marks background rectangle.

You can make Marks transparent by setting the Transparent property to False.

See Also

Chart Color property

TChart.Gradient

TChart.BackImage

See Also

TSeriesMarks.Frame

TSeriesMarks.Font

TSeriesMarks.Transparent

TSeriesMarks.Visible

BackImage Property

See also

Applies to

TChartcomponent

Declaration

```
property BackImage : TPicture;
```

Description

The BackImage property defines the Chart background image. You may define the image at design time.

You can use the BackImageMode property to control if the bitmap will be drawn "stretched", "normal" or "tiled".

See Also

TChart.BackImageInside

TChart.Color

TChart.Gradient

TChart.BackImageMode

BackImageInside Property

See also

cApplies to
TChart component

Declaration
```
property BackImageInside : Boolean;
```

Description
Default False

When True this property restricts display of the BackImage to inside the chart Frame area.

See Also

TChart.BackImage

TChart.BackImageMode

BackImageMode Property

See also

cApplies to
TChart component

Declaration
```
property BackImageMode : TTeeBackImageMode;
```

Description
Default pbmStretch

The BackImageMode property determines how will the background bitmap be displayed.

See Also
TChart.BackImage

BarWallRect Example

```
procedure TForm1.Series1BeforeDrawValues(Sender: TObject);
begin
 With Chart1.Canvas do
 begin
  Brush.Style:=bsSolid;
  Brush.Color:=clYellow;
  With Chart1.BackWallRect do Rectangle(Left,Top,Right,Bottom);
 end;
end;
```

BackWallRect Method

See also Example

Applies to

TChart, TDBChart components

Declaration

```
function BackWallRect:TRect;
```

Description

The BackWallRect function returns the Axis bounding rectangle (ChartRect) but adding the 3D percent offset in pixels to the Right and Top coordinates.

This is the rectangle coordinates of the back Chart wall.

You can use this rectangle to custom draw on Chart back wall space, using BeforeDrawValues event of any Series in the Chart.

See Also

TChart.ChartRect

TChart.Width3D

TChart.Height3D

BarBrush Example

This code will display Bars with patterns:

```
BarSeries1.BarBrush.Style := bsFDiagonal;
BarSeries1.BarBrush.Color := clWhite;
BarSeries1.SeriesColor  := clRed;
```

BarBrush Property

See also Example

Applies to

TBarSeries and THorizBarSeries components

Declaration

```
property BarBrush : TBrush;
```

Description

The BarBrush property defines the Brush used to fill Bars. You can set the Brush Color and Brush Style properties. When BarBrush.Style is different than bsSolid, the SeriesColor color is the background bar color.

See Also

TCustomBarSeries.BarPen

TCustomBarSeries.Dark3D

BarHeight Example

```
tmp := HorizBarSeries1.BarHeight;
```

BarHeight Property

See also Example

Applies to

THorizBarSeries component

Declaration

```
property BarHeight : Longint;
```

Description

Run-time and read only. The BarHeight property returns the height of horizontal Bars in pixels. Bar heights change at run-time when resizing or zooming the Chart. BarHeight is a read-only property. You can use the CustomBarHeight property to set a fixed pixels Bar height.

See Also

THorizBarSeries.CustomBarHeight

THorizBarSeries.SideMargins

TCustomBarSeries.BarWidthPercent

BarMargin Property

See also

Applies to

TBarSeries and THorizBarSeries components

Declaration

```
property BarMargin : LongInt;
```

Description

The BarMargin function returns the margin size in pixels. The SideMargins property controls if margins will be applied or not.

See Also
TCustomBarSeries.SideMargins

BarPen Property

See also

Applies to

TCustomBarSeries component

Declaration

```
property BarPen : TChartPen;
```

Description

The BarPen property indicates the kind of pen used to draw the Bar rectangles. You can set BarPen.Visible:=False to hide those lines.

See Also

TCustomBarSeries.BarBrush

TCustomBarSeries.BarStyle

BarStyle Property

See also Example

Applies to

TBarSeries, THorizBarSeries, TCustomBarSeries component

Declaration

```
property BarStyle : TBarStyle;
```

Description

Default Value: bsRectangle

The BarStyle property defines the Bar shape used to draw Bars.

BarStyle Property Example

This example sets the BarStyle to the Pyramid type.

```
BarSeries1.BarStyle := bsPyramid ;
```

See Also

TCustomBarSeries.BarWidthPercent

TCustomBarSeries.OffsetPercent

BarWidth Example

```
BarSeries1.CustomBarWidth := BarSeries1.BarWidth + 5 ;
```

BarWidth Property

See also Example

Applies to

TBarSeries component

Declaration

```
property BarWidth : LongInt;
```

Description

Run time and read only. The BarWidth property returns the width of vertical Bars in pixels. Bar widths change at run-time when resizing or zooming the Chart. BarWidth is a read-only property. You can use the CustomBarWidth property to set a fixed pixels Bar width.

See Also

TBarSeries.CustomBarWidth

TBarSeries.SideMargins

TCustomBarSeries.BarWidthPercent

BarWidthPercent Property

See also

Applies to

TCustomBarSeries component

Declaration

```
property BarWidthPercent : Integer;
```

Description

Default Value: 70

The BarWidthPercent property determines the percent of total Bar width used. Setting BarWidthPercent := 100 makes joined Bars. You can control how many Bars appear at same time by using TChart.MaxPointsPerPage property. The BarWidth and BarHeight properties indicate Bar dimensions in pixels.

See Also

TCustomBarSeries.OffsetPercent

TBarSeries.BarWidth

TBarSeries.CustomBarWidth

THorizBarSeries.BarHeight

THorizBarSeries.CustomBarHeight

OnBeforeDrawValues Event

Applies to
TChartSeries component

Declaration
```
property OnBeforeDrawValues : TNotifyEvent;
```

Description
BeforeDrawValues event is called just before the Series points will be painted.
You can then paint on the Chart.Canvas your customized drawings.

WARNING:
Do not change Series properties that would force the Series to be repainted.
This may cause recursion and endless repainting !

BottomAxis Property

See also

Applies to

TChart component

Declaration

```
property BottomAxis : TChartAxis;
```

Description

The BottomAxis property is one of the two horizontal chart axis. BottomAxis has many formatting properties and maintains the correct scales for Series X values. Each Series can be associated either to BottomAxis or TopAxis. Charts can have some Series associated to BottomAxis and some to TopAxis. Use the Minimum and Maximum properties to specify the scale and Increment property to control separation between axis labels. Axis Labels and scales can optionally be DateTime values. This is accomplished by setting MySeries1.XValues.DateTime := True.

Please refer to TChartAxis help for a complete explanation of properties.

See Also

TChart.TopAxis

TChart.DepthAxis

TChart.LeftAxis

TChart.RightAxis

BottomWall Property

See also

Applies to

TChart component

Declaration

```
property BottomWall : TChartWall;
```

Description

The BottomWall property defines the pen and brush used to fill the bottom chart side. Available TChartWall properties are Size, Color, Pen and Brush. TChart.View3D and TChart.View3DWalls properties should be True to use BottomWall , BackWall and LeftWall.

See Also

TChart.BackColor

TChart.BackWall

TChart.LeftWall

TChart.View3D

TChart.View3DWalls

Brush Property (TChartTitle)

See also

Applies to

TChartTitle component

Declaration

```
property Brush : TBrush;
```

Description

The Brush property determines the kind of brush used to fill the rectangle behind Chart.Title and Chart.Foot text.

See Also

TChartTitle.Frame

TChartTitle.AdjustFrame

Size Property (TChartWall)

See also

Applies to

TChartWall component

Declaration

```
property Size : Longint;
```

Description

The Size property determines the thickness of the selected Chart wall.

See Also

TChart.View3DWalls

TChartWall.Color

TChartWall.Brush

TChartWall.Pen

Brush Property (TChartWall)

See also

Applies to

TChartWall component

Declaration

```
property Brush : TBrush;
```

Description

The Brush property determines the kind of brush used to fill the Chart Walls background.

The Chart.View3DWalls property should be True to make walls visible.

Brush Property

See also

Applies to

TSeriesPointer component

Declaration

```
property Brush : TBrush;
```

Description

The Brush property determines the kind of brush used to fill Series Pointers.

See Also

TChart.View3DWalls

TChartWall.Color

TChartWall.Size

TChartWall.Pen

See Also
TSeriesPointer.Pen

BubbleCh Unit

The BubbleCh unit contains the declarations for the following components and for the enumerated type associated with them. When you add a component declared in this unit to a form, the unit is automatically added to the uses clause of that form's unit. The following items are declared in the this unit:

Components

TBubbleSeries

Types

Routines

To see a listing of items declared in this unit including their declarations, use the Project Browser.

BufferedDisplay Property

Applies to
TChart, TDBChart components

Declaration
```
property BufferedDisplay : Boolean;
```

Description
Default, BufferedDisplay is True.

Charts are drawn to an "internal" canvas and then it is shown on screen. This prevents "flickering" and, when charts have a lot of points, can gain speed as it's faster to show an entire chart than drawing all points directly to screen.

For real-time charts, where new points are continuously being added, this is a must.

CalcClickedPart Method

Applies to
TChart, TDBChart components

Declaration
```
Procedure CalcClickedPart (Pos:TPoint; Var Part:TChartClickedPart);
```

Description
```
TChartClickedPartStyle=  (  cpNone,
                cpLegend,
                cpAxis,
                cpSeries,
                cpTitle,
                cpFoot,
                cpChartRect );
TChartClickedPart=Record
   Part:TChartClickedPartStyle;
   PointIndex: Longint;
   ASeries:TChartSeries;
   AAxis:TchartAxis;
end;
```

CalcClickedPie Example

This code shows a message when a Pie Sector is clicked:

```
procedure TForm1.Chart1MouseDown(Sender: TObject; Button: TMouseButton;
 Shift: TShiftState; X, Y: Integer);
var tmp : Longint;
begin
 tmp := PieSeries1.CalcClickedPie( x, y );
 if tmp > -1 then
   ShowMessage( PieSeries1.XLabel[ tmp ] );
end;
```

CalcClickedPie Method

See also Example

Applies to

TPieSeries component

Declaration

```
function CalcClickedPie(x, y : Integer): LongInt;
```

Description

The CalcClickedPie function returns the Pie Sector Index that intersects with the X Y coordinate parameters. It currently works if Circled property is True.

See Also

TCircledSeries.PointToAngle

TCircledSeries.AngleToPos

CalcLabelStyle Method

See also

Applies to

TChartAxis component

Declaration

```
function CalcLabelStyle : TAxisLabelStyle;
```

Description

The CalcLabelStyle function returns the most logical Axis Label style. It calculates the "best candidate" label style based on how many Active Series are in the Chart and if the Series has point labels. The TChartAxis.LabelStyle property must be set to talAuto for this function to do its job.

If LabelStyle is not talAuto, the LabelStyle property value is returned.

See Also
TChartAxis.LabelStyle

CalcPosPoint Example

We want to calculate the LeftAxis value in the Vertical Screen coordinate of 220 pixels.

```
value := Chart1.LeftAxis.CalcPosPoint ( 220 ) ;
```

CalcPosPoint Method

See also Example

Applies to

TChartAxis component

Declaration

```
function CalcPosPoint(Value : Integer) : Double;
```

Description

This function returns the corresponding value of a Screen position. The Screen position must be between Axis limits.

See the user guide for more information.

See Also

TChartSeries.CalcXPos

TChartSeries.CalcYPos

CalcRect Method

See also

Applies to

TChartAxis component

Declaration

```
procedure CalcRect(Var R:TRect; InflateChartRectangle:Boolean);
```

Description

The CalcRect method determines the necessary space the Axis will need to display. It takes into account all Axis subcomponents dimensions like Title, Labels, Ticks, Width, etc.

The Rect parameter is modified with the amount in pixels.

See Also
TChart.ChartRect

CalcSize3D Method

See also

Applies to

TChart component

Declaration

```
procedure CalcSize3d;
```

Description

The CalcSize3D calculates the amount of pixels each Series needs to display in 3D mode. The TChart.View3D property must be True. The TChart SeriesWidth3d and SeriesHeight3d properties are modified. CalcSize3D is called internally before drawing.

See Also

TChart.CalcSize3DWalls

TChart.Width3d

TChart.Height3d

TChart.SeriesWidth3d

TChart.SeriesHeight3d

CalcSize3DWalls Method

See also

Applies to

TChart component

Declaration

```
procedure CalcSize3dWalls;
```

Description

The CalcSize3dWalls method calculates the amount of pixels of Chart Left Wall width and Bottom Wall height. The TChart.View3D property must be True. This method is called internally before drawing the Chart walls and before all Series points are displayed.

This method modifies the following TChart properties:

- Width3d
- Height3d
- SeriesWidth3d
- SeriesHeight3d

See Also

TChart.CalcSize3D

TChart.Width3d

TChart.Height3d

TChart.SeriesWidth3d

TChart.SeriesHeight3d

CalcXYIncrement Method

See also

Applies to
TChartAxis component

Declaration

```
function CalcXYIncrement(MaxLabelSize:Integer):Double;
```

Description

The CalcXYIncrement function returns the distance between axis labels expressed in axis scales.

This function applies to both Chart BottomAxis and TopAxis axis components.

Basically this function calculates the most appropriate Increment to make Axis labels ‘not overlapping’.

The TChartAxis.Increment property is used as a first increment to try. If Labels will overlap, a new and bigger increment is applied until labels fit on axis dimensions.

The TChartAxis.LabelsSeparation controls the minimum amount of allowed distance between axis labels.

This function is called internally so you'll need to use in very special situations.

See Also
TChartAxis.LabelsSeparation

CalcXPos Method

See also Example

Applies to

TChartSeries component

Declaration

```
function CalcXPos( ValueIndex : Longint ) : LongInt;
```

Description

Returns the pixel Screen Horizontal coordinate of the ValueIndex Series value.

This coordinate is calculated using the Series associated Horizontal Axis.

See the user guide for more information.

CalcXPos Example

We want to know which is the Horizontal Screen coordinate of the 4th Series point:

```
tmp := LineSeries1.CalcXPos( 3 ) ; (remember points start at 0)
```

See Also

TChart.CalcPospoint

TChartSeries.CalcYPos

CalcXPosValue Method

See also Example

Applies to

TChartAxis component

Declaration

```
function CalcXPosValue(Const Value : Double) : Longint;
```

Description

This function calculates the Horizontal coordinate in pixels of Value parameter.

See the user guide for more information.

CalcXPosValue Example

Our bottom axis has a Minimum of 20 and a Maximum of 1000.

We need to draw a vertical line that represents the value: 500.

```
x := Chart1.BottomAxis.CalcXPosValue( 500 ) ;
Chart1.Canvas.MoveTo ( x , Chart1.ChartRect.Bottom );
Chart1.Canvas.LineTo ( x , Chart1.ChartRect.Top );
```

See Also

CalcYPosValue function

CalcXSizeValue function

CalcYSizeValue function

CalcXPosValue Method

See also Example

Applies to

TChartSeries component

Declaration

```
function CalcXPosValue( Const Value : Double ) : Longint ;
```

Description

Returns the pixel Screen Horizontal coordinate of the specified Value.

This coordinate is calculated using the Series associated Horizontal Axis.

See the user guide for more information.

CalcXPosValue Method Example

We want to know which is the Horizontal Screen coordinate of 1234.56 value:

```
tmp := LineSeries1.CalcXPosValue( 1234.56 ) ;
```

CalcXSizeValue Method (TChartAxis)

See also

Applies to

TChartAxis component

Declaration

```
function CalcXSizeValue(Const Value : Double) : Longint;
```

Description

This function calculates the Horizontal SIZE in pixels of Value parameter.

See the user guide for more information.

CalcXSizeValue Method (TChartSeries)

See also Example

Applies to

TChartSeries component

Declaration

```
function CalcXSizeValue( Const Value : Double ) : Longint ;
```

Description

Returns the pixel Screen Horizontal DIMENSION (or size) of the specified Value.

This coordinate is calculated using the Series associated Horizontal Axis.

See the user guide for more information.

CalcXSizeValue Method (TChartSeries) Example

We want to know how much Screen pixels size corresponds to a 12.1 X value:

```
tmp := LineSeries1.CalcXSizeValue( 12.1 ) ;
```

That means tmp is the quantity of horizontal pixels an offset of 12.1 needs.

See Also

CalcYSizeValue function

CalcXPosValue function

CalcYPosValue function

CalcYPos Method (TChartSeries)

See also Example

Applies to

TChartSeries component

Declaration

```
function CalcYPos( ValueIndex : Longint ) : Longint ;
```

Description

Returns the pixel Screen Vertical coordinate of the ValueIndex Series value.

This coordinate is calculated using the Series associated Vertical Axis.

See the user guide for more information.

CalcYPos Method (TChartSeries) Example

We want to know which is the Vertical Screen coordinate of the 22nd Series point:

```
tmp := LineSeries1.CalcYPos( 21 ) ; (remember points start at 0)
```

See Also

TChart.CalcPospoint

TChartSeries.CalcYPos

See Also

CalcXPosValue function

CalcXSizeValue function

CalcYSizeValue function

CalcYPosValue Method

See also

Applies to

TChartAxis component

Declaration

```
function CalcYPosValue(Const Value : Double) : Longint;
```

Description

This function calculates the Vertical coordinate in pixels of Value parameter.

See the user guide for more information.

CalcYPosValue Method (TChartSeries)

Applies to
TChartSeries component

Declaration
```
function CalcYPosValue(Const Value : Double) : Longint;
```

Description
Returns the pixel Screen Vertical coordinate of the specified Value.

This coordinate is calculated using the Series associated Vertical Axis.

See the user guide for more information.

See Also

CalcXPosValue function

CalcYPosValue function

CalcXSizeValue function

CalcYSizeValue Method

See also

Applies to

TChartAxis component

Declaration

```
function CalcYSizeValue(Const Value : Double) : Longint;
```

Description

This function calculates the Vertical SIZE in pixels of Value parameter.

See the user guide for more information.

CalcYSizeValue Method (TChartSeries)

See also Example

Applies to

TChartSeries component

Declaration

```
function CalcYSizeValue(Const Value : Double) : Longint;
```

Description

Returns the pixel Screen Vertical DIMENSION (or size) of the specified Value.

This coordinate is calculated using the Series associated Vertical Axis.

See the user guide for more information.

CalcYSizeValue Method (TChartSeries) Example

We want to know how much Screen pixels size corresponds to a 5.32 Y value:

```
tmp := LineSeries1.CalcYSizeValue( 5.32 ) ;
```

That means tmp is the quantity of vertical pixels an offset of 5.32 needs.

CalcVisiblePoints Property

See also

Unit
TeEngine

Applies to
TChartSeries component

Declaration
```
property CalcVisiblePoints : Boolean;
```

Description
default is True

(Advanced)

The CalcVisiblePoints property determines if TChart will calculate the first and last visible points of a given Series. By default it's True, meaning TChartSeries.CalcFirstLastVisibleIndex method will calculate the first and last point index to draw.

When False, TChart forces the Series to draw all points. This can reduce speed when drawing zoomed or scrolled charts.

See Also

TChartSeries.AllowSinglePoint

TChartSeries.DrawBetweenPoints

TCustomSeries.ClickableLine

CalcZOrder Method

See also

Applies to

TChartSeries component

Declaration

```
procedure CalcZOrder; virtual;
```

Description

The CalcZOrder method calculates the "Z" Series position (ZOrder).

This is the order Series are drawn at 3D mode. Series with bigger ZOrder are drawn FIRST.

If Chart.View3D property is False, all Series will calculate their ZOrder as zero (no ZOrder).

This method is called internally.

The ApplyZOrder property forces, when False, to draw all Series at same ZOrder position.

See Also

TChartSeries.ZOrder

TChart.ApplyZOrder

TChart.MaxZOrder

CancelMouse property Example

```
procedure TGanttForm.Series1Click(Sender: TChartSeries; ValueIndex: Longint;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if Button=mbLeft then
  With sender do
    ValueColorValueIndex)
  else
    Chart1.CancelMouse:=False;
    {Setting CancelMouse to True notifies TeeChart Engine to STOP handling
  the mouseclick and NOT start zoom or scroll.}
end;
```

.

CancelMouse Property

See also Example

Unit

Chart

Applies to

All TeeChart components

Declaration

```
property CancelMouse : Boolean;
```

Description

Default: True

This public property substitutes the abort procedure call. If you wish the mouse to act with dual functionality, eg. Zoom or scroll with clickable series (see OnClickSeries event), then Cancelmouse should be used.

Set Cancelmouse to False in your mouse-handling code to make TeeChart respond to mouse clicks with zoom or scroll.

CanvasChanged Method

Applies to
TChart, TDBChart components

Declaration
```
Procedure CanvasChanged(Sender:TObject);virtual;
```

Description

Caption Property

See also

Applies to

TChartAxisTitle component

Declaration

```
property Caption : String;
```

Description

The Caption property defines the string of text used to draw near to each Chart Axis. When empty, no Title is displayed. Use the Angle and Font properties to control Axis Title formatting.

See Also

TChartAxis.Title

TChartAxisTitle.Font

ChangeHorizSize Method

See also

Applies to

TSeriesPointer component

Declaration

```
procedure ChangeHorizSize(NewSize : Integer);
```

Description

The ChangeHorizSize method modifies the current pointer horizontal size without forcing the Chart to repaint.

This is equivalent to use the TSeriesPointer.HorizSize property except it doesn't calls the Repaint method.

It is internally used by some series like TBubbleSeries and TCandleSeries.

See Also

TSeriesPointer.HorizSize

TSeriesPointer.ChangeVertSize

TChartSeries.Repaint

ChangeStyle Method

See also

Applies to

TSeriesPointer component

Declaration

```
procedure ChangeStyle(NewStyle : TSeriesPointerStyle);
```

Description

The ChangeStyle method modifies the Pointer style without forcing the Chart to repaint.

This is equivalent to use the TSeriesPointer.Style property except it doesn't calls the Repaint method.

See Also

TSeriesPointer.Style

TChartSeries.Repaint

ChangeVertSize Method

See also

Applies to

TSeriesPointer component

Declaration

```
procedure ChangeVertSize(NewSize : Integer);
```

Description

The ChangeVertSize method modifies the current pointer horizontal size without forcing the Chart to repaint.

This is equivalent to use the TSeriesPointer.VertSize property except it doesn't calls the Repaint method.

It is internally used by some series like TBubbleSeries and TCandleSeries.

See Also

TSeriesPointer.VertSize

TSeriesPointer.ChangeHorizSize

TChartSeries.Repaint

Chart Property

See also

Applies to

TQRChart component

Declaration

```
property Chart : TQRChart;
```

Description

The Chart property defines the TChart or TDBChart component to display on a TQRChart. TQRChart is an "interface" component. It must be associated to a TChart or TDBChart component using the Chart property.

Chart Scrolling and Panning

See also

TChart and TDBChart components allow, by default, runtime Scrolling. Users need to drag the mouse while holding the right mouse button pressed. You can deny Horizontal and / or Vertical scrolling by changing the Chart.AllowPanning property.

See the user guide for more information.

Example:

This will deny scrolling:

```
Chart1.AllowPanning := pmNone ;
```

You can also use: pmHorizontal, pmVertical, or pmBoth

Scrolling speed depends on:

- The number of Series and Series Points.
- The Chart Width and Height.
- The computer processor and Video card processor speed.
- The Video resolution and number of colors.
- The Windows version and the Video driver.
- The speed when dragging the mouse !

See Also

Chart Zooming

Chart Zooming by coding

Restoring Scroll and Zoom

TChart AllowPanning

Chart Unit

The Chart unit contains the declarations for the following components and for the enumerated type associated with them. When you add a component declared in this unit to a form, the unit is automatically added to the uses clause of that form's unit. The following items are declared in the this unit:

Components

TChart

TChartGradient

TChartWall

Types

AnimatedZoomFactor global variable

TChartClick

TChartClickAxis

TGradientDirection

TeeEraseBack

TeeScrollMouseButton global variable

TeeZoomMouseButton global variable

TTeeBackImageMode

Routines

GradientFill Method

To see a listing of items declared in this unit including their declarations, use the Project Browser.

Chart Zooming

See also

TChart enables to zoom at runtime by default. To Zoom a chart area, hold the left mouse button and drag mouse toward down/right. You'll see a rectangle around the selected area. Release the left mouse button to Zoom. You can continue zooming again and again.

To RESTORE (or UNDO) the zoom, drag a rectangle in the opposite direction (up/left).

TChart will rescale the axis Maximum and Minimum to show all dependent Series points. This is the default Zoom behavior. You can activate/deactivate this feature by toggling the Chart.AllowZoom boolean property.

See the user guide for more information.

Example:

```
{ this will disable runtime zoom }
Chart1.AllowZoom := False;
```

See Also

How to Zoom a chart area by coding?

Restoring Zoom and Scroll

Chart Scrolling

How to Zoom a chart area by coding?

See also

Example 1:

Zoom an area with "pixel" coordinates:

```
Rect.Left := 123 ;
Rect.Top  := 67 ;
Rect.Right := 175 ;
Rect.Bottom:= 100 ;
Chart1.ZoomRect( Rect );
```

Example 2:

Zoom an area with point value coordinates:

You need first to translate from value to pixel coordinates. To do so, you can use the Axis or Series components.

```
Rect.Left := LineSeries1.CalcXPosValue( 22.5 ) ;
Rect.Top  := LineSeries1.CalcYPosValue( 5000 ) ;
Rect.Right := LineSeries1.CalcXPosValue( 57.6 ) ;
Rect.Bottom:= LineSeries1.CalcYPosValue( 15000 ) ;
Chart1.ZoomRect( Rect );
```

See Also
TQRChart.TeePrintMethod

Chart3DPercent Property

See also

Applies to
TChart, TDBChart components

Declaration
```
property Chart3dPercent : Integer;
```

Description
Default value: 15

The Chart3DPercent property indicates the size ratio between Chart dimensions and Chart depth when Chart.View3D is True. You can specify a percent number from 1 to 100.

See Also

TChart.View3D

TChart.View3DWalls

TChart.SeriesWidth3D

TChart.SeriesHeight3D

TChartSeries.ZOrder

ChartBounds Example

This code draws a frame around a Chart component:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 With Chart1,Canvas do
 begin
  Brush.Style:=bsClear;
  Pen.Color:=clRed;
  With ChartBounds do Rectangle(Left,Top,Right,Bottom);
 end;
end;
```

ChartBounds Property

See also Example

Applies to

TChart component

Declaration

```
property ChartBounds : TRect;
```

Description

Run-time and read only. The ChartBounds property defines the rectangle dimension the last time the Chart was displayed. When drawing to Screen, it equals to the Left, Top, Width and Height properties. When drawing to Printer, it equals to the paper rectangle section where Chart is printed. You should always use the ChartBounds property. Never assume the Chart Left and Top coordinates are zero.

See Also

TChart.ChartRect

TChart.ChartWidth

TChart.ChartHeight

TChart.PrintPartial

TChart.PrintRect

See Also
TeeChart Gallery

ChartHeight Property

See also

Applies to

TChart, TDBChart component

Declaration

```
property ChartHeight : LongInt;
```

Description

Run-time and read only. The ChartHeight property indicates the total height in pixels between the Top and Bottom Chart Axis positions.

It is equal to:

```
ChartHeight:=ChartRect.Bottom-ChartRect.Top;
```

This IS NOT the same as Chart1.Height, because Chart margins, Axis margins, Legend width, etc are first subtracted from the Chart vertical size.

See Also

TChart.ChartRect

ChartWidth

ChartXCenter

ChartYCenter

ChartPreview Method

See also

Applies to

TChart, TDBChart component

Declaration

```
procedure ChartPreview(AForm: TForm; AChart: TChart);
```

Description

The new TChartPreview dialog allow Charts to be viewed on Screen matching as much as possible how they will look on the printer. The ChartPreview method (teeprevi.pas unit) shows a modal dialog with the specified Chart component. The end user can Print, change Printer properties, Paper Orientation and drag the Chart rectangle to any Paper position.

See Also

TChart.Print

TChart.PrintResolution

ChartPrintRect Method

See also

Applies to

TChart component

Declaration

```
function ChartPrintRect : TRect;
```

Description

The ChartPrintRect function returns the printer rectangle coordinates after PrintMargins have been applied. Coordinates are expressed in Printer device units (printer pixels).

See Also
TChart.PrintMargins

ChartRect Property

See also

Applies to

TChart component

Declaration

```
property ChartRect : TRect;
```

Description

Run-time and read only. The ChartRect property returns the screen coordinates of the chart axis bounding rectangle. That is, the rectangular space defined by the four Chart axis.

You must use ChartRect and ChartBounds when calculating custom drawing X Y positions. So, you should never assume Chart dimensions and origin are fixed values.

Example

Draws vertical stripes on the Chart background

```
procedure TDrawForm.LineSeries1BeforeDrawValues(Sender: TObject);
Const
  MyColors:array[1..5] of TColor=
  ( clNavy,
   clGreen,
   clYellow,
   clRed,
   $00000080 { very red }
   );
var t,partial:Longint;
  tmpRect:TRect;
  YPosition:Longint;
  tmpYCenterValue:Double;
begin
 With Chart1 do
 Begin
  { we will divide the total chart width by 5 }
  tmpRect:=ChartRect;
  tmpRect.Right:=tmpRect.Left;
  partial:=ChartWidth div 5;
  { change the brush style }
  Canvas.Brush.Style:=bsDiagCross;
  Canvas.Pen.Style:=psClear;
  { for each section, fill with a specific color }
  for t:=1 to 5 do
  Begin
   { adjust the rectangle dimension }
   tmpRect.Right :=tmpRect.Right+partial+1 ;
   { set the brush color }
   Canvas.Brush.Color:=MyColors[t];
   { paint !!! }
   With tmpRect do
    Canvas.Rectangle( Left+Width3D,Top-Height3D,Right+Width3D,Bottom-
  Height3D );
   { adjust rectangle }
   tmpRect.Left:=tmpRect.Right;
  end;
  { first calculate the middle vertical value (based on LineSeries points) }
  With LineSeries1.YValues do
```

```
     tmpYCenterValue:=MinValue+Percent*(MaxValue-MinValue)/100.0;
   { then calculate the Screen Pixel coordinate of the above value }
   YPosition:=LeftAxis.CalcYPosValue(tmpYCenterValue);
   With Canvas do
   begin
    { change pen and draw the line }
    Pen.Width:=3;
    Pen.Style:=psSolid;
    Pen.Color:=clRed;
    MoveTo(ChartRect.Left,YPosition);
    LineTo(ChartRect.Left+Width3D,YPosition-Height3D);
    LineTo(ChartRect.Right+Width3D,YPosition-Height3D);
   end;
  end;
end;
```

See Also

TChart.ChartBounds

TChart.ChartWidth

TChart.ChartHeight

TChart.ChartXCenter

TChart.ChartYCenter

TChart.Draw

ChartRegionRect Example

You can use it, for example, to check whether mouse cursor is inside a 3D chart:

```
if PtInRect( ChartRegionRect, Point( x,y ) ) then ...
```

ChartRegionRect Method

See also Example

Applies to

TChart component

Declaration

```
function ChartRegionRect : TRect;
```

Description

The ChartRegionRect is the bounding Chart axis rectangle:

Source:

```
Function TChart.ChartRegionRect:TRect;
Begin
 Result:=ChartRect;
 if View3D then
 Begin
  Inc(Result.Right,Width3d);
  Dec(Result.Top,Height3d);
 end;
end;
```

See Also

TChart.ChartRect

TChart.Height3D

TChart.Width3D

ChartWidth Property

See also

Applies to

TChart component

Declaration

```
property ChartWidth : LongInt;
```

Description

Run-time and read only. The ChartWidth property indicates the total width in pixels between the Left and Right Chart Axis positions.

It is equal to:

```
ChartWidth:=ChartRect.Right-ChartRect.Left;
```

This IS NOT the same as Chart1.Width, because Chart margins, Axis margins, Legend width, etc are first subtracted from the Chart horizontal size.

See Also

TChart.ChartHeight

TChart.ChartRect

ChartXCenter

ChartYCenter

ChartXCenter Method

See also

Applies to

TChart, TDBChart components

Declaration

```
function ChartXCenter : Longint;
```

Description

This function returns the middle Horizontal coordinate of the Chart.

This IS NOT the same as Chart1.Width / 2, because Chart margins, Axis margins, Legend width, etc are first subtracted from the Chart horizontal size.

i.e.: TPieSeries uses ChartXCenter for the Pie center.

See Also

ChartYCenter

ChartWidth

ChartYCenter Method

See also

Applies to

TChart, TDBChart components

Declaration

```
function ChartYCenter : Longint;
```

Description

This function returns the middle Vertical coordinate of the Chart.

This IS NOT the same as Chart1.Height / 2, because Chart margins, Axis margins, Legend width, etc are first subtracted to the Chart vertical size.

i.e.: TPieSeries uses ChartYCenter for the Pie center.

See Also

ChartXCenter

ChartWidth

SetFunction Method

Example

Applies to

TChartSeries component

Declaration

```
procedure SetFunction(AFunction:TTeeFunction); virtual;
```

Description

The SetFunction method allows you to add a Function definition as the Datasource of any Series.

CheckDataSource Method

Applies to
TChartSeries component

Declaration
```
procedure CheckDataSource;
```

Description
The CheckDataSource method will refresh all Series point values, either from database Tables, Queries or another Series points. You can call this method regularly if you want new or modified data to appear in realtime in the Series. The parent Chart will be repainted to reflect any changes.

CircleBackColor Property

See also

Applies to

TCircledSeries component

Declaration

```
property CircleBackColor : TColor;
```

Description

Default: clTeeColor

The CircleBackColor determines the color to fill the ellipse. Setting it to clTeeColor indicates the CircledSeries to use to Chart.Color color.

See Also

TCircledSeries.PiePen

TChart.Color

TChartGradient

CircleXCenter Property

See also

Applies to

TCircledSeries component

Declaration

```
property CircleXCenter : LongInt;
```

Description

Run-time and read only. The CircleXCenter public property returns the exact ellipse's center horizontal position in pixels.

The ellipse's radius is determined by CircledSeries.XRadius and YRadius properties. The AngleToPos function converts from angles to X and Y Screen coordinates. The PointToAngle function converts from XY Screen positions to angles.

See Also

TCircledSeries.CircleYCenter

TCircledSeries.XRadius

TCircledSeries.YRadius

TCircledSeries.AngleToPos

TCircledSeries.PointToAngle

CircleYCenter Property

See also

Applies to

TCircledSeries component

Declaration

```
property CircleYCenter : LongInt;
```

Description

Run-time and read only. The CircleYCenter public property returns the exact ellipse's center vertical position in pixels. The ellipse's radius is determined by CircledSeries.XRadius and YRadius properties. The AngleToPos function converts from angles to X and Y Screen coordinates. The PointToAngle function converts from XY Screen positions to angles.

See Also

TCircledSeries.CircleXCenter

TCircledSeries.XRadius

TCircledSeries.YRadius

TCircledSeries.AngleToPos

TCircledSeries.PointToAngle

Circled Property

See also

Applies to

TCircledSeries component

Declaration

```
property Circled : Boolean;
```

Description

Default Value: False

The Circled property defines if a CircledSeries such as TPieSeries will be drawn elliptically or with same X and Y radius (circle).

See Also

TCircledSeries.CircleBackColor

TCircledSeries.CustomXRadius

TCircledSeries.CustomYRadius

TCircledSeries.RotationAngle

TPieSeries

Clear Method

Applies to
TChartSeries component

Declaration
```
procedure Clear;
```

Description
This method deletes all Series values. Dependent Series are notified. If no new points are appended to the Series, nothing will be painted.

ClickableLine Property

See also

Unit
Series

Applies to
TCustomSeries component

Declaration
```
property ClickableLine : Boolean;
```

Description
default is True. (False in TPointSeries)

(Advanced)

The ClickableLine property determines if LineSeries accepts mouse clicks over the line drawn between points.

See Also

TChartSeries.AllowSinglePoint

TChartSeries.DrawBetweenPoints

TChartSeries.CalcVisiblePoints

Clicked Method

Applies to
TChartAxis component

Declaration
```
function Clicked(x,y: Integer) : Boolean;
```

Description
This function returns if X and Y coordinates are close to the Axis position.

Clicked Method

See also

Applies to

TChartSeries component

Declaration

```
function Clicked(x, y: Integer) : Longint; virtual;
```

Description

This functions returns the ValueIndex of the "clicked" point in the Series. Clicked means the X and Y coordinates are in the point screen region bounds. If no point is "touched", Clicked returns -1

See Also
OnClick

Clicked Method

See also

Applies to

TChartTitle component

Declaration

```
function Clicked(x, y: Integer) : Boolean;
```

Description

The Clicked method returns if mouse cursor is inside TChartTitle bound rectangle.

The Title.Visible property must be True.

The Title rectangle size depends on Title.Pen.Visible and Title.AdjustFrame properties.

See Also

TChartTitle.Frame

TChartTitle.AdjustFrame

Clip Property

Applies to
TSeriesMarks component

Declaration
```
property Clip : Boolean;
```

Description
The Clip boolean property toggles if Marks outside Chart limits will overwrite any other Chart region.

By default is True, meaning Marks will be drawn only within inner chart boundaries, keeping Axis Labels, Titles, Legend, etc almost untouched.

ClipPoints Property

Applies to
TChart, TDBChart components

Declaration
```
property ClipPoints : Boolean;
```

Description
ClipPoints boolean property toggles the drawing of Series points within Chart boundaries, preventing other Chart regions from being overwritten.

ClipRectangle Example

The following code creates a clipping rectangle around a Chart Legend component and draws a diagonal line. Try with and without clipping.

```
procedure TForm1.Chart1AfterDraw(Sender: TObject);
begin
 Chart1.ClipRectangle( Chart1.Legend.RectLegend ); { <-- comment this line }
 try
  With Chart1.Canvas do
  begin
   Pen.Style:=psSolid;
   Pen.Color:=clBlue;
   With Chart1.Legend.RectLegend do
   begin
    MoveTo( Left - 10, Bottom + 10 );
    LineTo( Right + 10, Top - 10 );
   end;
  end;
 finally
  Chart1.UnClipRectangle;
 end;
end;
```

ClipRectangle Method

See also Example

Applies to
TCanvas3D component

Declaration

```
Procedure ClipRectangle(Const Rect:TRect);virtual;
```

Description

The ClipRectangle method creates a Windows GDI clipping region and selects it into TChart.Canvas device context handle.

The UnClipRectangle method removes any clipping region applied to TChart.Canvas.

You can use this method to avoid Series or custom drawing to go over the desired rectangle coordinates.

Note: Printing and creating metafiles do not accept clipping regions in logical pixels.

See Also
TChart.Canvas.UnClipRectangle

Color Property

See also

Applies to

TChart, TDBChart components

Declaration

```
property Color : TColor;
```

Description

The Color property is the Color used to fill the complete Chart Panel background.

Color Property (TChartLegend)

Applies to
TChartLegend component

Declaration
```
property Color : TColor;
```

Description
Default Value: clWhite

The Color property defines the color used to fill Legend's background space.

Color Property (TChartPen)

Applies to
TChartPen component

Declaration
```
property Color : TColor;
```

Description
The Color property determines the color used by the pen to draw lines on the canvas.

The Color property can be any valid Delphi color constant like clRed, clGreen, clBtnText, etc.

A special color constant unique to TeeChart is: clTeeColor This is the "default color".

Each TeeChart drawing object has a different default color. For example, the TChart.Frame property has a default color of clBlack.

Color Property (TChartWall)

See also

Applies to

TChartWall component

Declaration

```
property Color : TColor;
```

Description

Default clTeeColor

The Color property specifies the color used to fill the Chart Walls background. The Chart.View3DWalls property should be True to make walls visible. Setting it to clTeeColor means TeeChart will use the Chart.BackColor.

See Also
Chart BackColor property

See Also

TChart.BackColor

TChart.View3DWalls

TChartWall.Brush

TChartWall.Pen

Color3D Property

See also

Applies to

TPieSeries component

Declaration

```
property Color3D : Boolean;
```

Description

Default Value: True

The Color3D property controls if the Pie 3D zone will be filled or not.

See Also

TPieSeries.ShadowColor

TPieSeries.Shadowed3d

ColorEachPoint Property

Applies to
TChartSeries component

Declaration
```
property ColorEachPoint : Boolean;
```

Description
Default value: False.

The TChartSeries ColorEachPoint property is a boolean property that controls which color will be drawn on the Series points. If False, all points will be drawn using the Series SeriesColor color property. If True, each Series point will be "colored" with its corresponding point color. The point colors are stored in the TChartSeries.ValueColor array property. If a point has a clTeeColor color value, then a palette color value will be used to draw it.

You can change this property both at design and runtime:
```
LineSeries1.ColorEachPoint := True ;
```

ColorPalette Constant Array Property

See also

Unit

TeEngine

Declaration

```
ColorPalette : Array[1..MaxDefaultColors] of TColor = (clRed, clGreen,
clYellow, clBlue, clWhite, clGray, clFuchsia, clTeal, clNavy, clMaroon,
clLime, clOlive, clPurple, clSilver, clAqua, clBlack);
```

Description

This global array of color constants is used in many TeeChart situations to obtain default color values for new added Series or to draw each Series point in a different color.

The GetDefaultColor function returns a specific color from ColorPalette array.

Note: This array is intended for read-only access, although you can customize your default colors by changing ColorPalette values in your application.

See Also
GetDefaultColor Global function

ColorRange Example

Imagine we have a LineSeries with ten years of data and we want the year 1993 to be in clBlue color:

Then we call:

```
LineSeries1.ColorRange( LineSeries1.XValues, EncodeDate(1993,1,1),
  EncodeDate(1993,12,31),
clBlue);
```

Imagine now we want all Y Values greater than 100 to be clYellow :

```
LineSeries1.ColorRange( LineSeries1.YValues, 100, LineSeries1.MaxYValue,
  clYellow);
```

ColorRange Method

See also Example

Applies to

TChartSeries component

Declaration

```
procedure ColorRange(AValueList: TChartValueList ;Const FromValue, ToValue:
Double; AColor: TColor);
```

Description

This method will change the Color of a specified range of points.

The FromValue and ToValue parameters are the beginning and end of the specified AValueList range.

AValueList can be any Series ValueList such as: XValues, YValues, etc.

ColorSource Property

Applies to
TChartSeries component

Declaration
```
property ColorSource : String;
```

Description
The ColorSource property must be a valid numeric Data Field Name.

TDBChart assigns every point's color to the ColorSource field value.

Color Values in Tables or Querys must be expressed as RGB values.

ColorWidth Property

See also

Applies to

TChartLegend component

Declaration

```
property ColorWidth : Integer;
```

Description

Default Value: 12

The ColorWidth property defines the width of Legend marks in percent of total Legend width. Each Series is shown in Chart Legend both with text and color mark. The color mark is a different shape for each different Series type.

See Also
TChartLegend.Color

ConnectingPen Property

See also

Applies to

TGanttSeries component

Declaration

```
property ConnectingPen : TChartPen;
```

Description

The ConnectingPen property determines the kind of pen used to draw the optional lines that connect Gantt Bars.

Gantt Bars are "connected" by using the TGanttSeries.NextTask property.

See Also
TGanttSeries.NextTask

CopyToClipboardBitmap Method

Applies to
TChart, TDBChart components

Declaration
```
Procedure CopyToClipboardBitmap;
```

Description
Copies the whole Chart area to clipboard in bitmap format.

CopyToClipboardMetafile Method

Applies to
TChart, TDBChart components

Declaration
```
Procedure CopyToClipboardMetafile(Enhanced:Boolean);
```

Description
Copies the whole chart area to the clipboard in metafile format. You may specify either Windows Metafile (WMF) or Enhanced Metafile (EMF) using the boolean operator.

```
Eg. Chart1.CopyToClipboardMetafile(True);
  { Enhanced Metafile = True }
```

Count Example

You can, for example, use this value to iterate:

```
for t := 0 to LineSeries1.Count - 1 do
LineSeries1.ValueColor [ t ] := clRed ;
```

Count Method

Example

Applies to

TChartSeries component

Declaration

```
function Count : Longint;
```

Description

This function returns the number of points in the Series.

Count Method

Example

Applies to

TChartValueList component

Declaration

```
function Count : Longint;
```

Description

This function returns the number of values in the List.

Count Method (TChartValueList) Example

You can, for example, use this function to locate a specific value:

```
for t := 0 to LineSeries1.YValues.Count - 1 do
if LineSeries1.YValue[ t ] = 1234.56 then
  LineSeries1.ValueColor[ t ] := clYellow ;
```

Cursor Example

You can also use the crTeeHand cursor image:

```
LineSeries1.Cursor:=crTeeHand ;
```

Cursor Property

See also Example

Applies to

TChartSeries component

Declaration

```
property Cursor : TCursor;
```

Description

Default: crDefault

The Cursor property is the image used when the mouse passes into the region covered by a Series point. Each Series determines the intersection of points with mouse coordinates each time the mouse moves. There are many different Cursors available. Refer to the Delphi help file. The Series ZOrder determines the order in which Series will be examined to calculate the clicked Series point.

See Also
TChartSeries.Clicked

CustomBarHeight Example

```
HorizBarSeries1.CustomBarHeight := HorizBarSeries1.BarHeight * 2 ;
```

CustomBarHeight Property

See also Example

Applies to
THorizBarSeries component

Declaration
```
property CustomBarHeight : Integer;
```

Description
Default: 0

The CustomBarHeight property determines the fixed Bar height in pixels. You can use the BarHeight property to get the current Bar height in pixels.

See Also

THorizBarSeries.BarHeight

THorizBarSeries.SideMargins

TCustomBarSeries.BarWidthPercent

CustomBarWidth Example

```
BarSeries1.CustomBarWidth := BarSeries1.BarWidth + 5 ;
```

CustomBarWidth Property

See also Example

Applies to

TBarSeries component

Declaration

```
property CustomBarWidth : Integer;
```

Description

Default: 0

The CustomBarWidth property determines the fixed Bar width in pixels. You can use the BarWidth property to get the current Bar width in pixels.

See Also

TBarSeries.BarWidth

TBarSeries.SideMargins

TCustomBarSeries.BarWidthPercent

CustomDraw method Example

Example draws 2 new axis

```
procedure TCustomAxisForm.LineSeries1AfterDrawValues(Sender: TObject);
var posaxis:longint;
begin
 With Chart1 do
 begin
  PosAxis:=ChartRect.Left+Trunc(ChartWidth*Percent/100.0);
  LeftAxis.CustomDraw(posaxis-10,posaxis-40,posaxis,DrawGrid.Checked);
  PosAxis:=ChartRect.Top+Trunc(ChartHeight*Percent/100.0);
  BottomAxis.CustomDraw(posaxis+10,posaxis+40,posaxis,DrawGrid.Checked);
 end;
end;
```

CustomDraw Method

See also Example

Applies to

TChartAxis component

Declaration

```
Procedure CustomDraw(PosLabels, PosTitle, PosAxis: LongInt; GridVisible: Boolean);
```

Description

The CustomDraw method displays an Axis at the specified screen positions with the current axis scales.

This method can be used in very special Charting applications, as it involves quite good TeeChart internals knowledge.

For normal charting the TChart component provides two horizontal and two vertical axis that are output to the screen automatically.

This method stores the Axis old positions, draws the Axis and restores the positions again.

The PosLabels, PosTitle and PosAxis parameters determine the Axis Labels, Title and Axis positions.

For horizontal Axis like TopAxis or BottomAxis, these positions are in vertical screen coordinates.

For vertical Axis like LeftAxis or RightAxis, these positions are in horizontal screen coordinates.

The GridVisible parameter determine if the Axis should draw or not the grid lines from the axis to chart edges.

The Axis is drawn using the current formatting properties such as fonts, ticks and colors.

See Also

TChartAxis.CustomDrawMinMax

TChartAxis.CustomDrawMinMaxStartEnd

TChartAxis.CustomDrawStartEnd

CustomDrawMinMax Method

See also

Applies to

TChartAxis component

Declaration

```
Procedure CustomDrawMinMax(PosLabels, PosTitle, PosAxis: LongInt;
GridVisible: Boolean; Const AMinimum, AMaximum, AIncrement: Double);
```

Description

The CustomDrawMinMax method displays an Axis at the specified screen positions with the specified axis scales.

This method can be used in very special Charting applications, as it involves quite good TeeChart internals knowledge.

For normal charting the TChart component provides two horizontal and two vertical axis that are output to the screen automatically.

This method stores the Axis old scales and positions, draws the Axis and restores everything again.

The PosLabels, PosTitle and PosAxis parameters determine the Axis Labels, Title and Axis positions.

For horizontal Axis like TopAxis or BottomAxis, these positions are in vertical screen coordinates.

For vertical Axis like LeftAxis or RightAxis, these positions are in horizontal screen coordinates.

The GridVisible parameter determine if the Axis should draw or not the grid lines from the axis to chart edges.

The AMinimum,AMaximum and AIncrement parameters define the Axis Minimum, Maximum and Increment properties.

The Axis is drawn using the current formatting properties such as fonts, ticks and colors.

See Also

TChartAxis.CustomDraw

TChartAxis.Increment

TChartAxis.LabelsSize

TChartAxis.Maximum

TChartAxis.Minimum

TChartAxis.PosAxis

TChartAxis.PosLabels

TChartAxis.PosTitle

TChartAxis.TitleSize

CustomDrawStartEnd Method

See also

Applies to

TChartAxis component

Declaration

```
Procedure CustomDrawStartEnd(APosLabels: Integer; APosTitle: Integer;
APosAxis: Integer; GridVisible: Boolean; AStartPos: Integer; AEndPos:
Integer);
```

Description

The CustomDrawStartEnd method displays an Axis at the specified screen positions with the current axis scales and defined Start / End positions. Using this method avoids the need to use the StartPosition and EndPosition properties.

See Also

TChartAxis.CustomDrawMinMax

TChartAxis.CustomDrawMinMaxStartEnd

TChartAxis.CustomDraw

TChartAxis.StartPosition

TChartAxis.EndPosition

CustomDrawStartEnd Method

See also

Applies to

TChartAxis component

Declaration

```
Procedure CustomDrawMinMaxStartEnd(APosLabels: Integer; APosTitle: Integer;
APosAxis: Integer; GridVisible: Boolean; Const AMinimum: Double; Const
AMaximum: Double; Const AIncrement: Double; AStartPos: Integer; AEndPos:
Integer);
```

Description

The CustomDrawStartEnd method displays an Axis at the specified screen positions with the current axis scales and defined Min / Max and Start / End positions. Using this method avoids the need to use the Axis Maximum, Minimum, StartPosition and EndPosition properties.

See Also

TChartAxis.CustomDrawMinMax

TChartAxis.CustomDrawStartEnd

TChartAxis.CustomDraw

TChartAxis.StartPosition

TChartAxis.EndPosition

CustomXRadius Property

See also

Applies to

TCircledSeries component

Declaration

```
property CustomXRadius : LongInt;
```

Description

Default 0

The CustomXRadius property indicates the amount in horizontal pixels used to calculate ellipse bounds. Setting it to zero means the CircledSeries will automatically calculate the adequate ellipse radius when necessary.

See Also

TCircledSeries.XRadius

TCircledSeries.YRadius

TCircledSeries.CustomYRadius

TCircledSeries.Circled

CustomYRadius Property

See also

Applies to

TCircledSeries component

Declaration

```
property CustomYRadius : LongInt;
```

Description

Default 0

The CustomYRadius property indicates the amount in vertical pixels used to calculate ellipse bounds. Setting it to zero means the CircledSeries will automatically calculate the adequate ellipse radius when necessary.

See Also

TCircledSeries.XRadius

TCircledSeries.YRadius

TCircledSeries.CustomXRadius

TCircledSeries.Circled

DBChart Unit

The DBChart unit contains the declarations for the following components and for the enumerated type associated with them. When you add a component declared in this unit to a form, the unit is automatically added to the uses clause of that form's unit. The following items are declared in the this unit:

Components

TDBChart

Types

Routines

To see a listing of items declared in this unit including their declarations, use the Project Browser.

Dark3D Property

See also

Applies to

TBarSeries and THorizBarSeries components

Declaration

```
property Dark3D : Boolean;
```

Description

Default True

The Dark3D property controls if bar sides will be filled with shadowed colors. This has effect only when Chart.View3D is True. High color video modes (greater than 256 colors) will show dark colors much better than 256 or 16 color modes.

Dark3D Property (TPieSeries)

See also

Applies to

TPieSeries component

Declaration

```
property Dark3D : Boolean;
```

Description

Default True

The Dark3D property indicates, when True, to fill the Pie 3D effect screen areas with darker colors than their corresponding Pie sectors. This colors look much better with 16k colors video mode or greater.

The Pie sector RGB color is increased 40 units to obtain the darker color.

See Also

TCustomBarSeries.BarBrush

TChart.View3D

TChart.IsScreenHighColor

See Also

TPieSeries.Shadowed3D

ApplyDark global function

DarkColor Property

See also Example

Applies to

TCustomBarSeries component

Declaration

```
property DarkColor : TColor;
```

Description

Run-time, read only. The DarkColor property returns the color used to fill the Bar sides or top cover. This applies both to TBarSeries and THorizBarSeries components. DarkColor and DarkerColor are used toghether to create a better visual effect.

First, the normal Bar color is converted to it's Red, Green and Blue values. Then, the normal Bar color is incremented by the following constant values:

```
Const DarkerColorQuantity=40;
   DarkColorQuantity =30;
```

Every Bar series style (Cyllinder, Pyramid, etc) will use DarkColor and DarkerColor for different visual parts.

For better visual results, you should install the video driver at a color resolution GREATER than 256 colors.

The TChart.IsScreenHighColor function returns if current video mode is a valid mode for better results.

You can change the above RGB constants to obtain lighter or darker results:

DarkColor property Example

```
DarkerColorQuantity:=20;
DarkColorQuantity:=10;
BarSeries1.Repaint;
```

See Also
TChart.IsScreenHighColor

DarkerColor Property

Applies to
TCustomBarSeries component

Declaration
```
property DarkerColor : TColor;
```

Description
Run-time, read only. The DarkerColor property returns the color used to fill the Bar sides or top cover. This applies both to TBarSeries and THorizBarSeries components. DarkColor and DarkerColor are used toghether to create a better visual effect.

First, the normal Bar color is converted to it's Red, Green and Blue values. Then, the normal Bar color is incremented by the following constant values:

```
Const DarkerColorQuantity=40;
   DarkColorQuantity =30;
```

Every Bar series style (Cyllinder, Pyramid, etc) will use DarkColor and DarkerColor for different visual parts.

For better visual results, you should install the video driver at a color resolution GREATER than 256 colors.

The TChart.IsScreenHighColor function returns if current video mode is a valid mode for better results.

You can change the above RGB constants to obtain lighter or darker results:

DataSource Property

Example

Applies to
TChartSeries component

Declaration
```
property DataSource : TComponent;
```

Description
Any TChartSeries can be optionally connected to a "point provider" (or DataSource). The "point provider" (or DataSource) can be:

1) Another Chart Series.

2) Any TTable, TQuery, TClientDataset or Delphi database dataset.

3) New ways of data feeding in the future. (WWW, ActiveForms, ActiveX...)

Points must be manually added by coding if no DataSource component is specified.

Please refer to Line, Bar, Point, Area, Bubble, Gantt,

DataSource property Example

{ This example shows how to connect a Series to a Table database component using DataSource property.}

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 { All of this can be done VISUALLY, using the Chart Editor Dialog:
 }
 { Do these 4 steps before:
  1) Place a TDBChart component and a TTable on the Form.
  2) Add a TBarSeries to DBChart1.
  3) Set Table1 to point to DBDEMOS database and ANIMALS table.
  4) Open Table1, setting Table1.Active to True.
 }
 Series1.DataSource:=Table1;       { <-- the Table component }
 Series1.YValues.ValueSource:='WEIGHT'; { <-- the Field for Bar Values }
 Series1.XLabelsSource:='NAME';     { <-- the Field for Bar Labels }
end;
```

Points must be manually added by coding if no DataSource component is specified.

Example of manual point adding:

```
Begin
  PieSeries1.Clear;
  for t := 1 to 12 do
    PieSeries1.Add( Random(1000), LongMonthNames[t], clTeeColor) ;
End;
```

DataSources Example

This code links 2 Series components to MySeries and sets MySeries to draw an Average on Series1 and Series2 points:

```
Uses TeeFunci ;
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
 With MySeries.DataSources do
 begin
  Clear;
  Add( Series1 );
  Add( Series2 );
 end;
 MySeries.SetFunction( TAverageTeeFunction.Create(Self) );
end;
```

DataSources Property

See also Example

Applies to

TChartSeries component

Declaration

```
property DataSources : TList read FDataSources;
```

Description

(Advanced)

The DataSources property stores component pointers which can be Series or DataSet components.

Any Series component can be linked to a DataSet (Table, Query, etc) or to one or more other Series components.

This standard TList property is handled internally so you will seldom use it directly.

Note: The TChartSeries.DataSource property returns the 0th element in DataSources list..

See Also
TChartSeries.DataSource

DateTime Example

```
LineSeries1.XValues.DateTime := False ; (we have X numbers, not DateTime X
  values)
BubbleSeries1.RadiusValues.DateTime := True ;
```

At runtime, you can manually set the values accordingly:

```
LineSeries1.XValues.Value[5]:=EncodeTime(23,59,00);
```

if LineSeries1.XValues.DateTime equals True.

Or...

```
LineSeries1.XValues.Value[5]:=1234.567;
```

if LineSeries1.XValues.DateTime equals False. However, no type checking is performed, so both ways will work.

See Also

TChartSeries.ExactDateTime

TChartAxis.DateTimeFormat

DateTime Property

See also Example

Applies to

TChartValueList component

Declaration

```
property DateTime : Boolean;
```

Description

TeeChart allows values to be expressed either as numbers or as Date+Time values. Delphi considers DateTime values as Doubles, thus making easier to handle both kinds of values. Each Series value list has a boolean property called DateTime. . The boolean DateTime property tells TeeChart what type the numbers are.The horizontal (x axis) and vertical (y axis) value defaults are number format (DateTime False). DateTime can be changed both at design-time and run-time, forcing the Chart to repaint. This property is used whenever a value must be converted to text, for example, to draw it as the chart axis labels. Axis labels will be drawn in DateTime or numeric format accordingly to the setting of the DateTime property.

You can also set the Chart Series ValueFormat and the Chart Axis DateTimeFormat formatting strings, to control how the values will be displayed.

DateTimeFormat Example

```
Chart1.BottomAxis.DateTimeFormat:='dd/mm/yy';
```

See Delphi help under FormatDateTime Function for complete details.

See Also

TChartAxis.Increment

TChartSeries.XValues

TChartSeries.YValues

TChartAxis.ExactDateTime

TChartValueList.DateTime

DateTimeFormat Property

See also Example

Applies to

TChartAxis component

Declaration

```
property DateTimeFormat : String;
```

Description

Chart Axis have a DateTimeFormat property. DateTimeFormat is a standard Delphi DateTime formatting string specifier. Chart Axis uses it to draw the axis labels.

Delete Example

We want to delete the 8th point of a Line Series:

```
LineSeries1.Delete( 7 ) ; (remember, points index start at zero)
```

This will remove completely the 8th point in the LineSeries1 Series. The chart will be repainted and all dependent Series (whom DataSource is LineSeries1) will be recalculated and redrawn.

Delete Method

Example

Applies to

TChartSeries component

Declaration

```
procedure Delete(ValueIndex : LongInt);
```

Description

The Delete method will remove the # ValueIndex point from the Series values lists. The ParentChart will be automatically redrawn. Dependent Series will be recalculated.

Delete Method (TChartValueList)

Applies to
TChartValueList component

Declaration
```
procedure Delete( ValueIndex : Longint);
```

Description
This method removes the specified ValueIndex value from the List.

DesignMaxPoints Property

Applies to
TChartSeries component

Declaration
```
property DesignMaxPoints : LongInt;
```

Description
The DesignMaxPoints integer property is the maximum number of points **in design mode** the Series will accept.

The default value is 200. Setting to 0 means there is no limit on the number of points. This property has no effect at runtime.

When you are dealing with DBChart components, and connecting Series to Tables or Querys, it can take a long time to retrieve and process all records. Setting DesignMaxPoints property to for example, 100, will reduce at design time the number of retrieved records to 100. That can speed design time development. Setting DesignMaxPoints to 0 will not limit the design time added points.

Direction Property

See also

Applies to

TChartGradient component

Declaration

```
property Direction : TGradientDirection;
```

Description

Default: gdTopBottom

The Direction property specifies the direction the gradient fill will be applied.

See Also

TChartGradient.EndColor

TChartGradient.StartColor

TChartGradient.Visible

DividingLines Property

Applies to
TChartLegend component

Declaration
```
property DividingLines : TChartHiddenPen read FDividingLines write
SetDividingLines;
```

Description
The DividingLines property specifies the Pen attributes used to draw lines separating Legend's items.

Lines are drawn horizontally for Left or Right aligned Legend and vertically for Top or Bottom Legend alignments..

DoSeriesClick Method

See also

Applies to

TChartSeries component

Declaration

```
procedure DoSeriesClick( ValueIndex:Longint; Button:TMouseButton; Shift:
TShiftState; X, Y: Integer); virtual;
```

Description

The DoSeriesClick method calls the Series OnClick event (if assigned).

Use the CancelMouse property to control how the mouse button behaves with dual modes (scroll or zoom after a Series OnClick event).

See Also
TChartSeries.OnClick

Draw Example

```
Printer.BeginDoc ;
Chart1.Draw( Printer.Canvas , MyRectangle ) ;
Printer.EndDoc ;
```

Where "MyRectangle" could be any valid rectangular region for the canvas.

```
MyRectangle.Left := 0 ;
MyRectangle.Right := Printer.PageWidth div 2 ;
MyRectangle.Top  := 0 ;
MyRectangle.Bottom:= Printer.PageHeight div 3 ;
```

Draw Method

Example

Applies to

TChart, TDBChart components

Declaration

```
procedure Draw ( UserCanvas : TCanvas ; Const UserRect : TRect ) ; virtual ;
```

Description

Use the Draw method to paint the Chart in your preferred Canvas and region.

It can also be used to Print the Chart in a customized way:

Draw Method (TChartAxis)

Applies to
TChartAxis component

Declaration
```
procedure Draw;
```

Description
This method paints the complete Axis (Ticks, Grid, Labels, Title) using the Axis.ParentChart.Canvas. Normally you do not need to call the Draw method directly.

Draw Method (TSeriesPointer)

See also

Applies to

TSeriesPointer component

Declaration

```
Procedure Draw(px, py : Integer; ColorValue : TColor; AStyle :
TSeriesPointerStyle );
```

Description

The Draw method displays a pointer at the specified px and py screen pixel coordinates.

The pointer is drawn using the current HorizSize and VertSize dimensions.

It calls the TSeriesPointer.DrawPointer method.

See Also
TSeriesPointer.DrawPointer

Draw3D Property

See also

Applies to

TSeriesPointer component

Declaration

```
property Draw3D : Boolean;
```

Description

Default Value: True

The Draw3D property controls if Series Pointers will be drawn in a 3D way or not. Currently only rectangle points have 3D capability.

See Also

TSeriesPointer.Style

TSeriesPointer.Brush

DrawArea Property

See also

Applies to

TCustomSeries component

Declaration

```
property DrawArea : Boolean;
```

Description

Default Value: False

The DrawArea property defines if bottom Chart side of points will be filled with a defined color. LineSeries and PointSeries set DrawArea to False. AreaSeries set DrawArea to True. You can control the Area Brush style by using the AreaBrush property.

See Also

TAreaSeries.SeriesColor

TAreaSeries.AreaBrush

TAreaSeries.AreaColor

TAreaSeries.AreaLinesPen

DrawBetweenPoints Property

See also

Unit
TeEngine

Applies to
TCustomSeries component

Declaration
```
property DrawBetweenPoints : Boolean;
```

Description
(Advanced)

The DrawBetweenPoints public property controls if Series needs to draw lines in the space between one point and another. TLineSeries sets DrawBetweenPoints to True.

It is internally used to calculate the first and last visible point. When True, TChart makes the Series to draw one more point to see the line between the last visible point and the next, if any.

See Also

TChartSeries.AllowSinglePoint

TCustomSeries.ClickableLine

TChartSeries.CalcVisiblePoints

DrawHorizontalLabel Method

Applies to
TChartAxis component

Declaration
```
procedure DrawHorizontalLabel(x,y:Longint; Angle:Integer; Const St:String);
```

Description
Horizontal Axis (TopAxis and BottomAxis) use this method to draw labels and Axis Title. Angle parameter must be an integer value between 0 and 360. X and Y are the text position coordinates. Text will be aligned to Left, Right, Top and / or Bottom depending on the Axis. Angles different than 0, 90, 180 or 270 may need manual XY calculation.

DrawPointer Method

See also

Applies to

TSeriesPointer component

Declaration

```
procedure DrawPointer(px, py, tmpHoriz, tmpVert : Integer; ColorValue :
TColor; AStyle : TSeriesPointerStyle);
```

Description

The Draw method displays a pointer at the specified px and py screen pixel coordinates with the tmpHoriz and tmpVert dimensions.

See Also
TSeriesPointer.Draw

DrawValuesForward Method

See also

Applies to

TChartSeries component

Declaration

```
function DrawValuesForward : Boolean; virtual;
```

Description

The DrawValuesForward function returns whether the Series needs to draw its points in ascending or descending order.

Some Series need to draw their points in descending order (starting from the last point to the first) depending on certain situations. For example, when the horizontal axis Inverted property is True.

See Also
TChartAxis.Inverted

DrawVerticalLabel Method

Applies to
TChartAxis component

Declaration
```
procedure DrawVerticalLabel(x,y: Longint; Angle: Integer; Const St: String);
```

Description
Vertical Axis (LeftAxis and RightAxis) use this method to draw labels and Axis Title. Angle parameter must be an integer value between 0 and 360. X and Y are the text position coordinates. Text will be aligned to Left, Right, Top and / or Bottom depending on the Axis. Angles different than 0, 90, 180 or 270 may need manual XY calculation.

Drawing Text with a Rotation Angle

Rotation Angle integer number can be 0, 90, 180, 270 or any other valid Angle between 0 and 359.
If you use a custom angle, margins will need to be calculated manually.

Orthogonal Property

See also

Applies to

TView3DOptions component

Declaration

```
property Orthogonal : Boolean;
```

Description

Disabling Orthogonal (setting to False) disables the 2D Canvas and enables the 3D Canvas allowing Elevation and Rotation displacement of the Chart.

See also

- HorizOffset
- Rotation
- Perspective
- Elevation
- Tilt
- VertOffset
- Zoom

Tilt Property

See also

Applies to

TView3DOptions component

Declaration

```
property Tilt : Integer;
```

Description

Default = 0

Tilt will rotate the Chart Rectangle within the Chart Panel. Positive values (from 0 to 360º) rotate the Chart anti-clockwise, negative values, clockwise.

***Important**. Orthogonal should be set to False for Rotation to act on the Chart.

See also

- Zoom
- Orthogonal
- Elevation
- Rotation
- HorizOffset
- VertOffset

VertOffset Property

See also

Applies to

TView3DOptions component

Declaration

```
property VertOffset : Integer;
```

Description

Default = 0

VertOffset will move the Chart Rectangle vertically across the Chart Panel. Positive values move the Chart downwards, negative values upwards.

See also

- Zoom
- Orthogonal
- Perspective
- Elevation
- Rotation
- Tilt
- HorizOffset

HorizOffset Property

See also

Applies to

TView3DOptions component

Declaration

```
property HorizOffset : Integer;
```

Description

Default = 0

HorizOffset will move the Chart Rectangle horizontally across the Chart Panel. Positive values move the Chart to the right, negative values to the left.

See also

- Zoom
- Orthogonal
- Elevation
- Perspective
- Rotation
- Tilt
- VertOffset

Zoom Property (View3DOptions)

See also

Applies to

TView3DOptions component

Declaration

```
property Zoom : Integer;
```

Description

Default = 100%

View3DOptions' Zoom will zoom the whole Chart. Expressed as a percentage, Increasing the value positively will bring the Chart towards the viewer, increasing the overall Chart size as the Zoom value increases. The Chart may be enlarged to a size greater than the Chart panel thus bringing the axes outside of the viewable area. View3DOptions' Zoom does not affect the use of Zoom. functionality either by code or mouse action, which may be used to zoom on Chart Series within the axes' boundaries.

Negative values of Zoom will diminish the overall Chartsize until, at values less than 0 the Chart will re-increase in size.

See also

- HorizOffset
- Orthogonal
- Elevation
- Perspective
- Rotation
- Tilt
- VertOffset
- ZoomText

Zoom Property (View3DOptions)

See also

Applies to

TView3DOptions component

Declaration

```
property ZoomText : Boolean;
```

Description

Default = True

ZoomText enables/disables the zooming of text when using the TView3DOptions.Zoom property.

See also

- HorizOffset
- Orthogonal
- Elevation
- Perspective
- Rotation
- Tilt
- VertOffset
- Zoom

Perspective Property

See also

Applies to

TView3DOptions component

Declaration

```
property Perspective : Integer;
```

Description

Perspective sets the view of the Chart (when orthogonal = False) with perspective effect (dimensional appearance with respect to distance from the viewer).

Values are integer from 0 -> 100%

When set to 0 perspective id disabled.

***Important**. Orthogonal should be set to False for Perspective to act on the Chart.

See also

- HorizOffset
- Orthogonal
- Elevation
- Rotation
- Tilt
- VertOffset
- Zoom

Rotation Property

See also

Applies to

TView3DOptions component

Declaration

```
property Rotation : Integer;
```

Description

Rotation describes front plane rotation by rotation degrees (0 - 360º). Increasing the value positively will bring the right of the Chart towards the viewer and the left of the Chart away, moving around a vertical axis at the central horizontal point of the Chart.

***Important**. Orthogonal should be set to False for Rotation to act on the Chart.

See also

- HorizOffset
- Orthogonal
- Elevation
- Perspective
- Tilt
- VertOffset
- Zoom

Elevation Property

See also

Applies to

TView3DOptions component

Declaration

```
property Elevation : Integer;
```

Description

Elevation describes front plane rotation by rotation degrees (0 - 360º). Increasing the value positively will bring the top of the Chart towards the viewer and the bottom of the Chart away, moving around an horizontal axis at the central vertical point of the Chart.

***Important**. Orthogonal should be set to False for Elevation to act on the Chart.

See also

- HorizOffset
- Orthogonal
- Perspective
- Rotation
- Tilt
- VertOffset
- Zoom

EndColor Property (TChartGradient)

See also

Applies to
TChartGradient component

Declaration

```
property EndColor : TColor;
```

Description
Default: clYellow

The EndColor property is one of the two colors used to create the gradient fill. The gradient fill is composed of two colors: StartColor and EndColor.

See also

TChartGradient.Direction

TChartGradient.StartColor

TChartGradient.Visible

EndValues Property

See also

Applies to

TGanttSeries component

Declaration

```
property EndValues : TChartValueList;
```

Description

The EndValues property defines the ending Gantt bar date value. The starting Gantt bar point is stored at TGanttSeries.StartValues

list property.

StartValues and EndValues can be specified both as DateTime or double values.

Both are standard TChartValueList components. That means you can access their values with same methods as you can access X or Y values.

The TGanttSeries.AddGantt and / or TGanttSeries.AddGanttColor methods must be used to add Gantt bar points.

See Also

TGanttSeries.AddGantt

TGanttSeries.AddGanttColor

TGanttSeries.StartValues

EndXValues Example

This code sets to 123 the X1 coordinate of 6th arrow point: (Remember point index starts at zero)

```
ArrowSeries1.EndXValues.Value[5]:= 123;
```

EndXValues Property

See also Example

Applies to

TArrowSeries component

Declaration

```
property EndXValues : TChartValueList;
```

Description

Each Arrow has (X0,Y0) and (X1,Y1) coordinates.

EndXValues property is the list of X1 values.

See Also

TChartValueList

TArrowSeries.EndYValues

TArrowSeries.StartXValues

TArrowSeries.StartYValues

TArrowSeries.AddArrow

EndYValues Example

This code modifies the Y1 coordinate of first arrow point:

```
tmp:=ArrowSeries1.EndYValues.Value[0];
ArrowSeries1.EndYValues.Value[0]:=tmp + 123;
```

EndYValues Property

See also Example

Applies to

TArrowSeries component

Declaration

```
property EndYValues : TChartValueList
```

Description

Each Arrow has (X0,Y0) and (X1,Y1) coordinates.

EndYValues property is the list of Y1 values.

See Also

TChartValueList

TArrowSeries.EndXValues

TArrowSeries.StartXValues

TArrowSeries.StartYValues

TArrowSeries.AddArrow

ExactDateTime Property

See also

Applies to
TChartAxis component

Declaration
```
property ExactDateTime : Boolean;
```

Description
Default Value: False

The ExactDateTime property controls if TChartAxis.Increment property calculates Axis Labels in exact DateTime steps.

This is very useful when TChartAxis.Increment is a DateTimeStep constant value:

```
Chart1.BottomAxis.Increment := DateTimeStep[ dtOneMonth ] ;
```

In this example, the Increment property should be considered to be an **exact month**. So, if an axis label is:

```
'1-April-1996'
```

then the next axis label would be:

```
'1-May-1996'
```

and the next:

```
'1-June-1996'
```

When ExactDateTime is False (the default value), the dtOneMonth increment equals to 30 days, and axis **do not** calculate how many days a month has.

The Series XValues or YValues properties should have the DateTime property True. (XValues for horizontal Axis and YValues for vertical Axis).

See Also

TChartAxis.Increment

TChartSeries.XValues

TChartSeries.YValues

TChartAxis.DateTimeFormat

TChartValueList.DateTime

ExchangeSeries Method

Applies to
TChart, TDBChart components

Declaration
```
Procedure ExchangeSeries(Series1,Series2 : Longint);
```

Description
This method changes the Series order, swapping one Series Z position with another. The Chart repaints to reflect the new Series order.
It accesses TChart.SeriesList property.

FillSampleValues Method

See also

Applies to

TChartSeries component

Declaration

```
procedure FillSampleValues(NumValues : Longint); virtual;
```

Description

Each Series draws random values at design mode unless you connect the Series component to other Series component or to a DataSet (Table,SQL,TClientDataset) component.

This method adds some random values to the Series and redraws it.

Example

```
Series1.FillSampleValues(20);
{will draw a series with 20 values at runtime}
```

See Also
NumSampleValues function

FillSequence Property

See also

Applies to
TChartValueList component

Declaration
```
procedure FillSequence;
```

Description
(Advanced)

The FillSequence method renumbers all values in a TChartValueList component. Values start at zero.

Warning: Calling FillSequence removes any previous value in a TChartValueList.

See TChartValueList.Sort method example.

See Also
TChartValueList.Sort

First Example

This is the same value as the 0 index value:

```
LineSeries1.XValues[ 0 ] := 1234.56 ;
ShowMessage( FloatToStr( LineSeries1.XValues.First ) );
```

First Method

Example

Applies to

TChartValueList component

Declaration

```
function First : Double;
```

Description

This function returns the First point value.

FirstValue Property

See also

Applies to
TChartLegend component

Declaration
```
property FirstValue : LongInt;
```

Description
Default Value: 0

The FirstValue property defines the number of the first displayed Legend item. Legend can display all active Series names or all points of a single Series.

FirstValue should be set accordingly, taking care not to overflow the number of active Series or the number of Series points.

You can use FirstValue to show in Legend a specific subset of Series or points.

It should be greater or equal than zero, and lower than number of active Series or Series points.

See TChartLegend.LegendStyle for a description of the different Legend styles.

See Also

TChartLegend.LegendStyle

TChartLegend.Inverted

FirstValueIndex Property

See also

Applies to

TChartSeries component

Declaration

```
property FirstValueIndex : LongInt;
```

Description

This function / property returns the ValueIndex of the First point in the Series that has an X value which is between the Horizontal Axis Maximum and Minimum values.

When no Zoom is applied to the Chart, and the Axis are Automatic, the FirstValueIndex is 0. If the Series has no points or there are no visible points, the FirstValueIndex returns -1.

See Also
LastValueIndex

Font Property (TChartAxisTitle)

See also

Applies to

TChartAxisTitle component

Declaration

```
property Font : TFont;
```

Description

The Font property determines the kind of font used to draw each Axis Title.

TChartAxis.Title.Caption property should be non empty.

Title text can be rotated using the Angle property.

Font Property (TChartLegend)

See also

Applies to

TChartLegend component

Declaration

```
property Font : TFont;
```

Description

The Font property determines the font used to draw all Legend text items. The Legend calculates its dimensions based on Font size and attributes as well as TChartLegend.ColorWidth and TChartLegend.Frame properties settings.

Font Property (TChartTitle)

Applies to
TChartTitle component

Declaration
```
property Font : TFont;
```

Description
The Font property determines the Chart Title and Footer.

Font Property (TSeriesMarks)

See also

Applies to

TSeriesMarks component

Declaration

```
property Font : TFont;
```

Description

The Font property determines the font used to draw the Series Marks.

The TSeriesMarks.Visible property must be True.

Series Marks are little text annotations close to each Series point.

See Also

TChartSeries.Marks

TSeriesMarks.BackColor

TSeriesMarks.Visible

See Also

TChartAxis.Title

TChartAxisTitle.Caption

See Also

TChartLegend.Color

TChartLegend.ColorWidth

TChartLegend.Frame

TChartLegend.LegendStyle

FontCanvas Method

See also

Applies to

TChart component

Declaration

```
procedure FontCanvas(SourceFont : TFont);
```

Description

The FontCanvas method assigns the SourceFont parameter to the Chart.Canvas.Font property. Internally, FontCanvas adjusts the Font PixelsPerInch property to override the default Delphi behaviour.

See Also

TChart.Canvas

TChart.PrintResolution

Foot Property

See also

Applies to

TChart component

Declaration

```
property Foot : TChartTitle;
```

Description

The Foot property defines the Text and formatting properties to be drawn at bottom Chart side. Use the Text property to enter the desired Foot lines, set Visible to True and change the Font, Frame and Brush properties. Use the Alignment property to control text output position.

See Also

TChart.Title

TChartTitle.Text

FormattedLegend Method

Applies to
TChart, TDBChart components

Declaration
```
function FormattedLegend (SeriesOrValueIndex : Longint) : String ;
```

Description
This function returns the text string corresponding to a Legend position.

The Legend position depends on the Legend.LegendStyle property. If LegendStyle is lsSeries, then the text string will be the SeriesOrValueIndexth Active Series Title. If LegendStyle is lsValues, then the text string will be the formatted SeriesOrValueIndexth value of the first Active Series in the Chart.If LegendStyle is lsAuto and only one Active Series exists in the Chart,then the LegendStyle is considered to be lsValues. If there's more than one Active Series then LegendStyle will be lsSeries.

Values are formatted accordingly to the LegendTextStyle property.

FormattedValue Example

This code shows the same text is used to display the Legend 5th item: (point's index starts at zero)

```
ShowMessage( Chart1.Legend.FormattedValue( LineSeries1, 4 ) );
```

FormattedValue Method

See also Example

Applies to

TChartLegend component

Declaration

```
function FormattedValue(ASeries : TChartSeries; ValueIndex : Longint) : String;
```

Description

This function returns the corresponding Legend text for the Series ValueIndex point. The Legend.LegendTextStyle property is used to properly format the point values and labels.

See Also

TChartSeries.ValueFormat

TChartSeries.PercentFormat

TChartSeries.XLabel

FormattedValueLegend Method

See also

Applies to

TChart component

Declaration

```
function FormattedValueLegend( ASeries: TChartSeries; ValueIndex: Longint): String;
```

Description

The FormattedValueLegend function returns the string representation of a Series Point value just as it would appear in Chart.Legend.

The ValueIndex parameter is the point index.

The Legend.TextStyle property and all other TChartLegend properties are used to create the resulting string.

See Also
TChartLegend.TextStyle

Frame Example

This code sets the Chart frame:

```
Chart1.Frame.Visible := True ;
Chart1.Frame.Color := clBlue ;
Chart1.Frame.Width := 1 ;
Chart1.Frame.Style := psDot ;
```

See Also

TChart.ChartRect

TChart.View3D

TChart.BackColor

TChartGradient

TChartWall

TChartTitle.AdjustFrame

TChartTitle

TChartTitle.Brush

Frame Property (TChartLegend)

See also

Applies to
TChartLegend component

Declaration

```
property Frame : TChartPen;
```

Description
The Frame property determines the kind of pen used to draw a frame around Legend rectangle.

Frame Property

See also Example

Applies to

TChart and TChartTitle component

Declaration

```
property Frame : TChartPen
```

Description

The Frame property indicates the Pen used to draw a frame around the Chart axis. The frame is displayed using the Chart.ChartRect coordinates. The TChartPen also defines the TeeChart Title Frame and is used throughout the TeeChart components to define Pen characteristics.

Frame Property (TSeriesMarks)

See also

Applies to

TSeriesMarks component

Declaration

```
property Frame : TChartPen;
```

Description

The Frame property determines the kind of pen used to draw a rectangle around a Series Marks.

See Also
TChartLegend.Color

See Also

TSeriesMarks.BackColor

TSeriesMarks.Font

TSeriesMarks.Transparent

TSeriesMarks.Visible

GanttCh Unit

The GanttCh unit contains the declarations for the following components and for the enumerated type associated with them. When you add a component declared in this unit to a form, the unit is automatically added to the uses clause of that form's unit. The following items are declared in the this unit:

Components

TGanttSeries

Types

Routines

To see a listing of items declared in this unit including their declarations, use the Project Browser.

GetASeries Method

See also

Applies to

TChart component

Declaration

```
function GetASeries : TChartSeries;
```

Description

The GetASeries function returns the FIRST Active Series in a Chart. If Chart has no Series, or none of the Series is Active, this function returns NIL.

See Also

TChart.Series

TChart.SeriesList

TChartSeries.Active

GetAxisSeries Method

Applies to
TChart, TDBChart components

Declaration
```
function GetAxisSeries ( Axis : TChartAxis ) : TChartSeries ;
```

Description
This function returns the first Series that depends on the specified Axis. If no Series depend on Axis, the nil value is returned.

GetBarStyle Method

See also

Applies to
TCustomBarSeries component

Declaration

```
function GetBarStyle(ValueIndex: LongInt) : TBarStyle;
```

Description

The GetBarStyle function returns the corresponding Bar Style for a given "ValueIndex" Bar.

The ValueIndex parameter must be a valid bar index in the range of zero to Count - 1.

Normally Bar Styles are the same for all Bar points, but you can use the OnGetBarStyle event to supply a different Bar style for each Bar point.

This function calls the OnGetBarStyle if assigned.

See Also

TCustomBarSeries.BarStyle

TCustomBarSeries.OnGetBarStyle

GetColorRect Method

See also

Applies to

TChartLegend component

Declaration

```
function GetColorRect(X1,Y0,Y1:Longint):TRect;
```

Description

The GetColorRect function returns the rectangle coordinates used to draw the color mark near to Legend items.

This function needs X origin and Y origin and ending coordinates and calculates the X ending position based on Legend.ColorWidth property value.

.

See Also
TChartLegend.ColorWidth

GetCursorPos Method

Applies to
TChart, TDBChart components

Declaration
```
function GetCursorPos:TPoint;
```

Description
The GetCursorPos function returns a TPoint record containing current mouse cursor position coordinates in pixels. It calls Windows GetCursorPos and adjusts coordinates to TChart origin position using ScreenToClient method.

GetCursorValueIndex Example

This code displays the point Label text under the mouse cursor:

```
procedure TForm1.Chart1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  tmp : Longint;
begin
  tmp := PieSeries1.GetCursorValueIndex;
  if tmp > -1 then
  ShowMessage( PieSeries1.XLabel[ tmp ] );
end;
```

GetCursorValueIndex Method

See also Example

Applies to

TChartSeries component

Declaration

```
function GetCursorValueIndex : LongInt;
```

Description

The GetCursorValueIndex function returns the Series point index which is close to the mouse Screen coordinates. It returns -1 if there's no point close to the mouse cursor position.

See Also

TChartSeries.Cursor

TChartSeries.Clicked

TChartSeries.GetCursorValues

GetCursorValues Method

See also

Applies to

TChartSeries component

Declaration

```
procedure GetCursorValues( Var x , y : Double );
```

Description

The GetCursorValues method returns the corresponding Series X and Y values where mouse cursor is over. The X and Y values correspond to the Series associated Axis scales. One example of this can be found at teedemo.dpr example project under "Cross-Hair" (ucrossh.pas) unit.

See Also

TChartSeries.GetCursorValueIndex

TChartSeries.Cursor

TChartSeries.Clicked

GetDefaultPattern Method

See also

Applies to
All TeeChart Components

Declaration
```
function GetDefaultPattern(PatternIndex : Longint): TBrushStyle;
```

Description
The GetDefaultPattern function returns a valid Windows Brush Style from an internal palette of 6 possible Brush styles. The PatterIndex parameter is reduced to one of this 6 possible cases.

See Also

TPieSeries.UsePatterns

TChartLegend.PatternColors

GetEditorClass Example

The GetEditorClass function is now a "Class" function and can be called directly without creating any instance:

```
Var tmp : String;
tmp := TLineSeries.GetEditorClass;
instead of:
With TLineSeries.Create(Self) do
try
 tmp := GetEditorClass;
finally
 Free;
end;
```

GetEditorClass Method

See also Example

Applies to
TChartSeries component

Declaration

```
class function TChartSeries.GetEditorClass : String;
```

Description
This function returns the class name of the Series Editor Dialog. The default editor dialog classes are:

Series	Editor Class Name	Unit File Name
TChartSeries	'TChartSeriesEditor'	SERIEDIT
TCustomSeries	'TCustomSeriesEditor'	CUSTEDIT
TCustomBarSeries	'TBarSeriesEditor'	BAREDIT
TPieSeries	'TPieSeriesEditor'	PIEEDIT

Warning:
Series Editor Dialogs must be REGISTERED in the initialization Unit Section:

```
initialization
  RegisterClass ( TCustomSeriesEditor ) ;
end.
```

GetFreeSeriesColor Method

See also

Applies to

TChart component

Declaration

```
function GetFreeSeriesColor( CheckBackground : Boolean ) : TColor;
```

Description

The GetFreeSeriesColor function returns a color from the default color palette not used by any Series. The CheckBackGround parameter controls if the returned color should or shouldn't be the Chart BackColor color. This function returns a Color which is not used by any Series in the Chart.

See Also

ColorPalette

IsFreeSeriesColor

GetHorizAxis Example

You can use this to change the Axis properties:

```
LineSeries1.GetHorizAxis.LabelsOnAxis := False ;
```

GetHorizAxis Method

See also Example

Applies to

TChartSeries component

Declaration

```
function GetHorizAxis : TChartAxis;
```

Description

Every Series is associated either to the Top or Bottom Chart Horizontal Axis. This function returns this "associated" Axis.

See Also
GetVertAxis

GetLabelsSeries Method

See also

Applies to
TChart component

Declaration

```
function GetLabelsSeries(Axis : TChartAxis) : TChartSeries;
```

Description

The GetLabelsSeries function returns the Series component associated with the Axis parameter.

Each Series is associated with an horizontal and vertical axis through the TChartSeries.HorizAxis and VertAxis properties.

The Axis components call this function to get the correct Series component to draw all Labels.

If no Series exist, or if no Series is associated to the Axis parameter, nil is returned.

See Also

TChartAxis.LabelStyle

TChartSeries.HorizAxis

TChartSeries.VertAxis

GetMarkValue Method

See also

Applies to

TChartSeries component

Declaration

```
function GetMarkValue(ValueIndex : LongInt) : Double;
```

Description

The GetMarkValue function returns the corresponding Point value suitable for displaying at Series Marks. Most Series return the YValue[ValueIndex] value, but some special Series like THorizBarSeries return the XValue as Axis are inverted. Calling this virtual method assures receiving the proper value.

See Also

TChartSeries.ValueMarkText

TChartSeries.XValue

TChartSeries.YValue

TChartSeries.Marks

GetOriginPos Method

See also

Applies to

TAreaSeries component

Declaration

```
function GetOriginPos(ValueIndex : LongInt) : Longint;
```

Description

The GetOriginPos function returns the vertical point coordinate in Screen pixels for a given Area point origin value. The MultiBar property determines if origin is the Chart.ChartRect.Bottom coordinate or if it's the PointOrigin for Stacked and Stacked 100% styles.

GetOriginPos Method

See also

Applies to

TBarSeries and THorizBarSeries component

Declaration

```
function GetOriginPos(ValueIndex : Longint) : Longint;
```

Description

The GetOriginPos function returns the corresponding screen pixels coordinate of the leftmost horizontal bar edge.

When MultiBar property is different than Stacked or Stacked 100%, this function returns the YOrigin screen coordinate.

When several bar Series are stacked, this function returns a different screen coordinate for each Series.

The ValueIndex parameter specifies the bar index.

See Also

TAreaSeries.MultiArea

TChartSeries.PointOrigin

See Also

TCustomBarSeries.MultiBar

TCustomBarSeries.YOrigin

TCustomBarSeries.UseYOrigin

GetRectangle Method

See also

Applies to

TChart, TDBChart components

Declaration

```
function GetRectangle:TRect; virtual;
```

Description

The GetRectangle function returns the TChart panel bounding rectangle coordinates. It is virtual as it's used by TQRChart component to calculate the exact coordinates when previewing and printing Charts in QuickReport printable components. You should use GetRectangle when custom drawing on QuickReport Printer canvas. TChart and TDBChart simply return GetClientRect rectangle coordinates.

See Also

TChart.ChartBounds

TChart.ChartRect

TChart.GetWidthHeight

GetVertAxis Method

See also

Applies to

TChartSeries component

Declaration

```
function GetVertAxis:TChartAxis;
```

Description

Every Series is associated either to the Left or Right Chart Vertical Axis. This function returns this "associated" Axis. You can use this to change the Axis properties:

```
BarSeries1.GetVertAxis.AxisValuesFormat := '#,##0.0#' ;
```

See Also
GetHorizAxis

GetWidthHeight Method

See also

Applies to

TChart, TDBChart components

Declaration

```
Procedure GetWidthHeight(Var tmpWidth,tmpHeight:Longint);
```

Description

The GetWidthHeight method returns TChart panel's width and height using TChart.GetRectangle coordinates. It is equivalent to GetRectangle (Right - Left) and (Bottom - Top) values.

See Also
TChart.GetRectangle

GetYValueList Method

Applies to
TChartSeries component

Declaration
```
function GetYValueList(AListName:String):TChartValueList; virtual;
```

Description
All Series types have at least the XValues and YValues ValueLists. Some Series have more ValueLists. For example, CandleSeries have the "Close", "High" and "Low" valuelists. You can access these ValueLists by name using the above function:

tmp := CandleSeries1.GetYValueList('HIGH').MaxValue ;

This can be useful if you're doing generic functions.

Gradient Property

See also

Applies to

TChart component

Declaration

```
property Gradient : TChartGradient;
```

Description

The Gradient property specifies the colors used to fill Chart background. Chart background is filled using these two colors: StartColor and EndColor. You can control the drawing output by setting the TChartGradient.Direction property. Use the Visible property to show / hide filling.

See Also

TChart.BackColor

TChart.Color

TChart.BackImage

TChartGradient.Direction

GradientFill Example

This code fills a Form background:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  GradientFill( Canvas, ClientRect, clYellow, clBlue, True);
end;
```

GradientFill Method

See also Example

Applies to
TCanvas

Declaration

```
procedure GradientFill(Canvas: TCanvas; Const Rect: TRect; top, bottom:
TColor; Horizontal: Boolean);
```

Description
The GradientFill method is used to fill a Screen area with multi-colored lines to obtain a nice shadow effect and coloured backgrounds. The Chart.Gradient component uses this method internally. Chart.pas must be in your uses clause.

See Also
TChart.Gradient

Grid Property

See also

Applies to

TChartAxis component

Declaration

```
property Grid : TChartPen;
```

Description

The Grid property determines the kind of pen used to draw the Axis lines at every Axis Label position. These are the lines which go from "cousin Axis" Minimum to "cousin Axis" Maximum screen positions at each of our Label's position.

See Also

TChartAxis.Axis

TChartAxis.LabelsFont

TChartAxis.MinorTicks

TChartAxis.OnGetAxisLabel

TChartAxis.Ticks

TChartAxis.TicksInner

GridPen Property

See also

Applies to

TChartAxis component

Declaration

```
property GridPen : TChartPen
```

Description

The GridPen axis property indicates the kind of pen used to draw axis Grid lines. Left and Right axis draw horizontal grid lines, while Top and Bottom axis draw vertical grid lines.

Warning:

Set GridPen.Width := 0 when printing with non solid Pen Styles.

See Also

TChartAxis.Ticks

TChartAxis

Height3D Example

This code draws a vertical "frame":

```
With Chart1,Canvas do
begin
  MoveTo( ChartRect.Left + 10 , ChartRect.Bottom ) ;
  LineTo( ChartRect.Left + 10 , ChartRect.Top ) ;
  LineTo( ChartRect.Left + 10 + Width3D , ChartRect.Top - Height3D ) ;
end;
```

Height3D Property

See also Example

Applies to

TChart component

Declaration

```
property Height3D : Longint
```

Description

Run-time and read only. The Height3D property determines the height in pixels of the 3D effect. The Chart.View3D property should be True. It equals zero if Chart.View3D is False.

TChart.SeriesHeight3D
TChart.SeriesWidth3D
TChart.Width3D

HorizAxis Example

You can change the desired Horizontal Axis both at design and runtime:

```
BubbleSeries1.HorizAxis := aTopAxis ;
```

HorizAxis Property

See also Example

Applies to

TChartSeries component

Declaration

```
property HorizAxis : THorizAxis;
```

Description

The HorizAxis property is of type: THorizAxis. It means by which Horizontal Axis (Top or Bottom axis) will be the Series horizontally scaled.

See Also
TChartSeries.VertAxis

HorizMargin Property

See also

Applies to

TChartLegend component

Declaration

```
property HorizMargin : Integer;
```

Description

Default Value: 0

The HorizMargin property speficies the number of screen pixels between Legend and Chart rectangles. By default its 0, meaning Legend will calculate a predefined margin based on total Legend width.

It's only used when Legend position is Left or Right aligned.

See Also

TChartLegend.VertMargin

TChart.MarginLeft

TChart.MarginRight

HorizSize Property

See also

Applies to

TSeriesPointer component

Declaration

```
property HorizSize : Integer;
```

Description

Default Value: 4

The HorizSize property specifies the Series Pointer width in logical pixels.

Series that derive from TPointSeries usually override the HorizSize and VertSize properties.

For example, TBubbleSeries uses the Radius property to determine the correct HorizSize and VertSize, so these properties have no effect in that Series.

See Also

TSeriesPointer

TSeriesPointer.VertSize

Horizontal Property

See also

Applies to

TChartAxis component

Declaration

```
property Horizontal : Boolean;
```

Description

Run-time and read only. This property returns true if the Axis is horizontal. Chart.TopAxis and Chart.BottomAxis are horizontal. Chart.LeftAxis and Chart.RightAxis are vertical.

Chart Axis	Horizontal	OtherSide
LeftAxis	False	False
TopAxis	True	True
RightAxis	False	True
BottomAxis	True	False

See Also
Axis.OtherSide function

How to create charts with manually inserted values

Data Aware Charts

If values are manually inserted it means that you do not want TeeChart to automatically retrieve the Series points from a Table or Query.

1) Place a TChart component in a Delphi Form.

2) Add a Series component via the Chart Editor. (For example, you can choose a BarSeries component.)

3) In your Form1.OnCreate event (or in a Button1.OnClick event), type the following code:

```
Series1.Clear;  {Where Series1 is your Bar series}
Series1.Add( 25 , 'Barcelona'   , clTeeColor );
Series1.Add( 50 , 'Rome'     , clTeeColor );
Series1.Add( 30 , 'San Francisco' , clTeeColor );
Series1.Add( 50 , 'New York'   , clTeeColor );
Series1.Add( 40 , 'Los Angeles'  , clTeeColor );
Series1.Add( 35 , 'London'    , clTeeColor );
```

That's it.

Now lets add a new Line Series component (Series2):

4) Place a Line Series component in the Form.

5) Add the following code to the previous code:

```
Series2.Clear;  {Where Series2 is your LineSeries}
Series2.Add( 25 , '' , clTeeColor );
Series2.Add( 50 , '' , clTeeColor );
Series2.Add( 30 , '' , clTeeColor );
Series2.Add( 50 , '' , clTeeColor );
Series2.Add( 40 , '' , clTeeColor );
Series2.Add( 35 , '' , clTeeColor );
```

(Notice you do not need to insert the Horizontal Labels again).

clTeeColor is a TeeChart color constant that means "draw the point in the default color". You can assign a specific color (clRed, clBlue, etc) to each point or you can set the LineSeries1.ColorEachPoint property to True in order to automatically draw each point with a different color.

If you have X (Horizontal) Coordinates:

Replace the AddY method with AddXY:

```
Series2.AddXY( 15.2 , 25.4 , 'Barcelona' , clBlue );
```

Other Chart Series types:

Other Series types may use the same methods to add points:

```
PieSeries1.Add( 333 , 'Sales' , clTeeColor );
```

Whilst some series that rely on more inputs have series specific methods to add points:

```
BubbleSeries1.AddBubble( 15.2 , 25.4 , 13 , '' , clRed ); {being 13 the
  bubble radius}
```

How to Create Data-Aware Charts

Manual Charts

The only difference between TChart and TDBChart is TChart does **not** need the Borland Database Engine.

However, points must be manually added by programming.

TDBChart, in contrast, accepts Series connected to database components (like TTable, TQuery or TClientDataset).

TDBChart will retrieve database records and will automatically add the points.

Add a data table to your form

1) Place a TTable component and assign to it an existing Table. It is not necessary to make the table Active at this stage.

Steps to create a simple data-aware chart

(You can do all steps without programming, using the Chart editor)

2) Place a TDBChart in your form.

3) Right mouse key on the Chart to call up the TeeChart menu then select the Chart editor menu option or Double-click on the chart to go straight to the Chart editor.

4) In the Chart editor add a Series to the Chart by clicking on the **Add** button. This will call the TeeChart Gallery.Select a series type and click **OK**.

5) Back in the Chart editor select the Series page with the tab selector (or double-click on your series in the editor series list)

6) On the series page select your new series in the Series Combobox (If you only have one series in the chart it will automatically be displayed in the combobox). Click on the datasource tab to add the new datasource.

7) Select Dataset from the dropdown combobox to define the DataSource property.

Now you can set the series data sources:

8) Set the **Dataset** field to Table1.

9) Now you need to set Table Fields that will be used to draw your data series: Set the Labels field to the desired table Field. (for example: Table1.CityName) Set the Y values to the desired Vertical Table Field. (for example: Table1.Population). The fields available in this tab page of the Chart editor will depend on the series you are adding. For example a LineSeries will a permit the use of X and Y co-ordinates and an optional Label field, a Pie-series only uses the label field and one set of values. See the Series unit for more information.

Activate your data

10) Now set Active to True (or Open) the Table. Opening or closing the Table will force TDBChart to retrieve the records again, thus allowing refresh.

Color in fields:

It is also possible to have the point colors in a Table or Query field. The field must be a Numeric field containing RGB (Red,Green,Blue) values. Assign the LineSeries1.ColorSource property to this field.

Colors are expressed as numbers, please refer to the Delphi and Windows help to know more about RGB colors. (TColorType)

See Also

Chart Zooming

Restoring Zoom and Scroll

Increment Example

```
Chart1.BottomAxis.Increment := DateTimeStep[ dtOneHour ] ;
Chart1.RightAxis.Increment := 1000 ;
```

Increment Property

See also Example

Applies to

TChartAxis component

Declaration

```
property Increment : Double;
```

Description

Axis Increment is the minimum step between axis labels. It must be a positive number or DateTime value. TChart will use this value as the starting axis labels step. If there is not enough space for all labels, TChart will calculate a bigger one. You can use the DateTimeStep constant array for DateTime increments.

See Also

Axis Maximum

Axis Minimum

InflateMargins Property

See also

Applies to

TSeriesPointer component

Declaration

```
property InflateMargins : Boolean
```

Description

The InflateMargins property controls if Chart dimensions will be rescaled to accomodate the Series.Pointer HorizSize and VertSize pixels. When False, Chart Axis scales will be preserved and points close to the Axis limits will be partially displayed.

See Also

TSeriesPointer.HorizSize

TSeriesPointer.VertSize

Inverted Property

See also

Applies to

TChartAxis component

Declaration

```
property Inverted : Boolean;
```

Description

Default False

When Inverted is True, Axis Minimum and Maximum scales will be swapped. Axis labels and Series points will be displayed accordingly. This applies both to vertical and horizontal axis.

Inverted Property (TChartLegend)

See also

Applies to

TChartLegend component

Declaration

```
property Inverted : Boolean;
```

Description

Default False

The Inverted property indicates, when True, to draw the Legend items in opposite direction. Legend strings are displayed starting at top for Left and Right Aligment and starting at left for Top and Bottom Legend orientations. You can use the Legend.FirstValue property to determine the ValueIndex for the first Legend text item.

See Also

TChartAxis.Maximum

TChartAxis.Minimum

See Also

TChartLegend.Alignment

TChartLegend.FirstValue

InvertedStairs Property

Applies to
TLineSeries TAreaSeries component

Declaration
```
property InvertedStairs : Boolean;
```

Description
This boolean property controls the LineSeries or AreaSeries drawing.

When Stairs is set to True you may set InvertedStairs to True to alter the direction of the step. - see Stairs.

In most normal situations, the Series draws a line between each Line point. This makes the Line appear as a "mountain" shape. However, setting Stairs to TRUE will make the Series draw 2 Lines between each pair of points, thus giving a "stairs" appearance. This is most used in some financial Chart representations. You may invert the stair by setting InvertedStairs to true.

IsDateTime Example

```
if Chart1.BottomAxis.IsDateTime then...
```

IsDateTime Method

Example

Applies to

TChartAxis component

Declaration

```
function IsDateTime : Boolean;
```

Description

Each Chart Axis can consider values to be normal numbers or DateTime values. This function returns if the Axis dependent values are DateTime or not.

IsFreeSeriesColor Example

This code sets the MySeries1.SeriesColor to the clBlue color only if clBlue is not used by any other Series or the Chart BackColor:

```
if Chart1.IsFreeSeriesColor ( clBlue , True ) then
  MySeries1.SeriesColor := clBlue ;
```

IsFreeSeriesColor returns if AColor is NOT used by any Chart.Series.

```
If Chart1.IsFreeSeriesColor( clRed ) then
  LineSeries1.SeriesColor := clRed ;
```

IsFreeSeriesColor Method

See also Example

Applies to

TChart component

Declaration

```
function IsFreeSeriesColor(AColor: TColor; CheckBackground: Boolean) :
Boolean;
```

Description

The IsFreeSeriesColor returns whether the AColor parameter is used by any Series or not. The CheckBackGround parameter controls if AColor will be checked against the Chart.BackColor or not.

See Also

ColorPalette

GetFreeSeriesColor

IsScreenHighColor Example

This code shows or hides the Chart.Gradient:

```
Chart1.Gradient.Visible := Chart1.IsScreenHighColor ;
```

IsScreenHighColor Method

See also Example

Applies to

TChart component

Declaration

```
function IsScreenHighColor : Boolean;
```

Description

The IsScreenHighColor function returns True if the current video mode color depth is greater than 256 colors.

See Also
WarningHighColor

IsValidDataSource Method

See also

Applies to

TChart component

Declaration

```
function IsValidDataSource(ASeries: TChartSeries; AComponent: TComponent):
Boolean; virtual;
```

Description

The IsValidDataSource function determines if the AComponent parameter is a valid component to provide the point values for the ASeries component.

The TChart component accepts TChartSeries components as valid DataSources. The TDBChart component accepts TChartSeries and any TDataSet (TTable, TQuery, TClientDataset, etc) components as valid DataSources.

This function is used at design-time and run-time to control Series have valid point values sources.

See Also
TChartSeries.DataSource

IsValidSeriesSource Method

Applies to
TChartSeries component

Declaration
```
function IsValidSeriesSource(Value : TChartSeries): Boolean; virtual;
```

Description
The IsValidSeriesSource function checks if the Value series parameter is a valid candidate series point provider.

When TChartSeries.DataSource property is assigned to another Series, all points inserted in that Series are automatically propagated to all other associated Series.

This function is currently used only in TRSIFunction component. This special Series type allows only one specific Series type as DataSource: a TOHLCSeries or TCandleSeries.

All other Series components will return True.

IsValidSourceOf Method

Applies to
TChartSeries component

Declaration
```
function IsValidSourceOf(Value: TChartSeries): Boolean; virtual;
```

Description
The IsValidSourceOf function returns False if the Value parameter is the same as Self.
It's used to validate the DataSource property both at design and run-time.

LabelHeight Method

See also

Applies to

TChartAxis component

Declaration

```
function LabelHeight(Const Value: Double): LongInt;
```

Description

This function returns the Axis Label height of the Value parameter. It uses the Axis formatting specifiers, the Axis Labels Font and the Labels rotation and style.

See Also
LabelWidth

LabelStyle Property

See also

Applies to

TChartAxis component

Declaration

```
property LabelStyle: TAxisLabelStyle;
```

Description

Setting the Axis.LabelStyle property to talAuto will force the Axis to guess what labels will be drawn.

For each Active associated Series, if the Series have XLabels then the LabelStyle will be talText.

If no Series have XLabels, then LabelStyle will be talValue.

If no Active Series are associated with the Axis, the LabelStyle will be talNone.

See Also
Chart.OnGetAxisLabel event

LabelValue Method

See also

Applies to

TChartAxis component

Declaration

```
function LabelValue(Const Value: Double): String;
```

Description

This function returns the corresponding text representation of the Value parameter. It uses the Axis formatting specifiers.

See Also

LabelHeight

LabelWidth

LabelWidth Method

See also

Applies to

TChartAxis component

Declaration

```
function LabelWidth(Const Value : Double) : Longint;
```

Description

This function returns the Axis Label width of the Value parameter. It uses the Axis formatting specifiers, the Axis Labels Font and the Labels rotation and style.

See Also
LabelHeight

Labels Property

See also

Applies to

TChartAxis component

Declaration

```
property Labels : Boolean;
```

Description

Default Value: True

The Labels property shows or hide Axis Labels. Set it to False to draw the Axis Ticks and / or Grid lines only.

Use the LabelsAngle and LabelStyle properties to control Label appearing.

Use the Visible property to show or hide both the Axis Labels, Ticks and Grid lines.

See Also

TChartAxis.Grid

TChartAxis.LabelsAngle

TChartAxis.LabelsFont

TChartAxis.LabelStyle

TChartAxis.OnGetAxisLabel

TChartAxis.Ticks

TChartAxis.Visible

LabelsAngle Property

See also

Applies to

TChartAxis component

Declaration

```
property LabelsAngle : Integer;
```

Description

Default Value: 0

The LabelsAngle property defines the rotation degree applied to each Axis Label. Valid degree angle values are 0, 90, 180, 270 and 360. Some printers and video drivers fail when drawing rotated fonts or calculating the rotated font dimensions. Metafile Charts containing rotated fonts sometimes place text at sligthly different coordinates.

See Also

TChartAxis.Grid

TChartAxis.Labels

TChartAxis.LabelsFont

TChartAxis.LabelStyle

TChartAxis.OnGetAxisLabel

TChartAxis.Ticks

TChartAxis.Visible

LabelsFont Property

See also

Applies to

TChartAxis component

Declaration

```
property LabelsFont : TFont;
```

Description

The LabelsFont property determines the Font used to draw Axis Labels. The default Font is Arial. TrueType fonts look much better and perform more exactly. The LabelsFont.Size determines the number of non-overlapping axis labels.

See Also

TChartAxis.Labels

TChartAxis.LabelsAngle

TChartAxis.LabelStyle

TChartAxis.Grid

TChartAxis.OnGetAxisLabel

TChartAxis.Ticks

TChartAxis.Visible

LabelsOnAxis Property

See also

Applies to

TChartAxis component

Declaration

```
property LabelsOnAxis : Boolean;
```

Description

Default Value: True

The LabelsOnAxis property controls if Labels just at Axis Minimum and Maximum positions will be shown or NOT.

Set it to False, for example, to remove the "0" zero axis Label when Axis scales are from 0 to 1001:

```
Chart1.LeftAxis.SetMinMax( 0, 1001 );
Chart1.LeftAxis.LabelsOnAxis := False;
```

When Chart.View3D is True, Axis scales can vary to adapt Series minimum and maximum points dimensions.

See Also

TChartAxis.Labels

TChartAxis.LabelsFont

TChartAxis.LabelStyle

TChartAxis.LabelsAngle

TChartAxis.OnGetAxisLabel

LabelsSeparation Property

See also

Applies to
TChartAxis component

Declaration
```
property LabelsSeparation : Integer;
```

Description
Default Value: 10

The LabelsSeparation property specifies the percent amount of minimum distance between Axis Labels.

Setting it to "0" zero makes TChartAxis skip calculating overlapping labels. (No clipping is performed).

Labels visibility depends also on LabelsFont size, LabelsAngle and Axis.Increment properties.

See Also

TChartAxis.Labels

TChartAxis.LabelsFont

TChartAxis.LabelStyle

TChartAxis.LabelsAngle

TChartAxis.OnGetAxisLabel

LabelsSize Example

You can set this property both at design or runtime:

```
Chart1.RightAxis.LabelsSize := 50 ; (50 pixels separation)
```

LabelsSize Property

Example

Applies to

TChartAxis component

Declaration

```
property LabelsSize : Integer;
```

Description

The LabelsSize property is 0 by default.

Therefore the space between the Axis and the Chart will be automatically calculated based on the Axis Labels Width and Height.

Last Example

This is the same value as the Count - 1 index value:

```
LineSeries1.YValues[ LineSeries1.YValues.Count - 1 ] := 1234.56 ;
ShowMessage( FloatToStr( LineSeries1.YValues.Last ) );
```

Last Method

Example

Applies to

TChartValueList component

Declaration

```
function Last : Double;
```

Description

This function returns the Last point value. This is the same value as the Count - 1 index value:

LastValueIndex Property

See also

Applies to

TChartSeries component

Declaration

```
property LastValueIndex : Longint;
```

Description

This property returns the ValueIndex of the Last point in the Series that has an X value which is between the Horizontal Axis Maximum and Minimum values.

When no Zoom is applied to the Chart, and the Axis are Automatic, the LastValueIndex is the same as TChartSeries.Count-1.

If the Series has no points or there are no visible points, the LastValueIndex returns -1.

See Also
FirstValueIndex

DepthAxis Property

See also

Applies to

TChart component

Declaration

```
property DepthAxis : TChartAxis;
```

Description

The DepthAxis property determines the Labels and formatting attributes of Depth Chart axis (Z axis). It also controls where Series points will be placed.

Every TChart component has five TChartAxis: Left, Top, Right, Bottom and Depth (Z).

Refer to TChartAxis help topic for a complete description.

See Also

TChartAxis

TChart.LeftAxis

TChart.TopAxis

TChart.RightAxis

TChart.BottomAxis

TChart

LeftAxis Property

See also

Applies to

TChart component

Declaration

```
property LeftAxis : TChartAxis;
```

Description

The LeftAxis property determines the Labels and formatting attributes of Left Chart side. It also controls where Series points will be placed.

Every TChart component has five TChartAxis: Left, Top, Right, Bottom and Depth (Z).

The LeftAxis is pre-defined to be:

```
Horizontal := False ;
OtherSide := False ;
```

Refer to TChartAxis help topic for a complete description.

See Also

TChartAxis

TChart.DepthAxis

TChart.TopAxis

TChart.RightAxis

TChart.BottomAxis

TChart

BackWall Property

See also

Applies to

TChart component

Declaration

```
property BackWall : TChartWall;
```

Description

The BackWall property determines the drawing attributes of Chart Back wall. Chart.View3D and Chart.View3DWalls properties must be

set to TRUE for BackWall to be shown.

You can use the Color, Size, Pen and Brush TChartWall's properties to control the wall appearance.

A Chart component has also a BottomWall and LeftWall properties.

See Also

TChartWall

TChart.BottomWall

TChart.LeftWall

LeftWall Property

See also

Applies to

TChart component

Declaration

```
property LeftWall : TChartWall;
```

Description

The LeftWall property determines the drawing attributes of Chart Left side. Chart.View3D and Chart.View3DWalls properties must be

set to TRUE for LeftWall to be shown.

You can use the Color, Size, Pen and Brush TChartWall's properties to control the wall appearance.

A Chart component has also a BottomWall and BackWall properties.

See Also

TChartWall

TChart.BackWall

TChart.BottomWall

Legend Property

See also

Applies to

TChart component

Declaration

```
property Legend : TChartLegend;
```

Description

The Legend property determines the text and drawing attributes of Chart's textual representation of Series and Series values.

The TChartLegend component draws a rectangle and for each Series in a Chart (or for each point in a Series) outputs a text representation of that Series (or that point).

You can use the Legend.LegendStyle and Legend.TextStyle properties to control the text used to draw the legend.

The Legend can be positioned at Left, Right, Top and Bottom chart sides using the Legend.Alignment property.

Use the Legend.Visible property to show / hide the Legend.

The Inverted property makes Legend to draw text starting from bottom.

The Frame, Font and Color properties allow you to change Legend appearance.

The Legend.ColorWidth property determines the percent width of each item's "colored" mark.

The Legend.FirstValue property controls which Series (or Series point) will be used to draw first Legend item.

See Also

TChart

TChartLegend

LegendStyle Property

See also

Applies to

TChartLegend component

Declaration

```
property LegendStyle : TLegendStyle;
```

Description

Default Value: lsAuto

The LegendStyle property defines which items will be displayed in Chart Legend.

lsSeries style shows the TChartSeries.Title of all active Series in a Chart. Whenever a Series Title is empty, the Series Name property is used.

lsValues style shows a text representation of all points of the first active Series in a Chart.

lsLastValues style shows the last point value and the TChartSeries.Title of all active Series in a Chart. It is useful for real-time charting, where new points are being added at the end of each Series.

lsAuto style (the default) means LegendStyle will be lsSeries when there's more than one Active Series, and lsValues when there's only one Series in a Chart.

The TChartLegend.TextStyle property determines how the Series point values are formatted.

See Also

TChartLegend.TextStyle

TChartSeries.Active

LineBrush Property

See also

Applies to

TLineSeries component

Declaration

```
property LineBrush : TBrushStyle;
```

Description

Default Value: bsSolid

The LineBrush property defines the brush style used to fill LineSeries contents.

It has effect only when Chart1.View3D is True.

The Legend reflects automatically the selected brush style.

Setting different brush styles can be useful when printing on monochrome printers.

See Also
TLineBrush.LinePen

LinePen Example

This code changes FastLineSeries's pen style:

```
FastLineSeries1.LinePen.Style := psDot;
```

LinePen Property

See also Example

Applies to

TAreaSeries and TFastLineSeries components

Declaration

```
property LinePen : TChartPen
```

Description

The LinePen property determines what kind of pen will be used for drawing the line connecting all points.

See Also
TChartPen

LinkedSeries Property

Applies to
TChartSeries component

Declaration
```
property LinkedSeries : TList read FLinkedSeries;
```

Description
(Advanced)

The LinkedSeries public property is a standard Delphi TList component that stores all Series components linked to the Series. Series can be linked to other Series by using TeeChart gallery Functions or setting the DataSource property directly. All Series maintain a list of dependent Series to notify on point value changes to allow them to recalculate and redraw. TeeChart uses this list internally so you will seldom need to access it directly.

Locate Method

Applies to
TChartValueList component

Declaration
```
function Locate (Const Value:Double):Longint;
```

Description
This new function returns the corresponding point index which has the specified “Value”. You can use it to calculate X co-ordinates based on Y values or vice-versa:

```
tmp:=LineSeries1.XValues.Locate(EncodeDate(1996,1,1));
if tmp<>-1 then
  ShowMessage(FloatToStr(LineSeries1.YValues.Valuetmp));
```

Logarithmic Property

Applies to
TChartAxis component

Declaration
```
property Logarithmic : Boolean;
```

Description
This boolean property scales the Axis Logarithmically when True. Axis Minimum and Maximum values should be greater than 0, and Axis cannot be of DateTime type.

MandatoryValueList Example

This code shows the most important value for a Series 5th point:

(remember points start at zero)

```
ShowMessage( FloatToStr( LineSeries1.MandatoryValueList.Value[ 4 ] ) );
```

MandatoryValueList Method

See also Example

Applies to

TChartSeries component

Declaration

```
function MandatoryValueList : TChartValueList;
```

Description

The MandatoryValueList function returns the Series ValueList that stores the most important Series values. Most Series types return the YValues list, but some return the XValues list (THorizBarSeries, for example).

See Also

TChartSeries.ValueList

TChartValueList

Margin Properties

Example

Applies to

TChart, TDBChart components

Declaration

```
property MarginLeft : Integer;
property MarginTop : Integer;
property MarginRight : Integer;
property MarginBottom : Integer;
```

Description

Each Chart component has four margin properties: LeftMargin, RightMargin, TopMargin, BottomMargin.

These properties are expressed in screen pixels. Default values are 8 for top and bottom margins and 12 for left and right margins.

Margins Example

You can change both at design and runtime the margins properties:

```
Chart1.LeftMargin := 60 ;
```

MarkPercent Example

This will show the 0 Point Y Value in percent format: 12.23%

```
ShowMessage( LineSeries1.MarkPercent( 0, False ) );
```

This adds the total to the end of the string: 12.23% of 55400

```
ShowMessage( LineSeries1.MarkPercent( 0, True ) );
```

This function uses the PercentFormat formatting string property. When AddTotal is TRUE, then the global string variable PercentOf is used to separate the percent text from the total text. By default, PercentOf string is: " of ".

You can change it to the desired text:

```
PercentOf := ' from ' ;
```

This string constant is 32 char length.

MarkPercent Method

Example

Applies to

TChartSeries component

Declaration

```
function MarkPercent(ValueIndex: Longint; AddTotal: Boolean): String;
```

Description

This function returns a textual representation of a Point Value. Series components use this function internally to paint Marks in percent format. This function uses the PercentFormat formatting string property. When AddTotal is TRUE, then the global string variable PercentOf is used to separate the percent text from the total text. By default, PercentOf string is: " of ".

MarkText Method

Applies to
TChart, TDBChart components

Declaration
```
function MarkText ( ASeries : TChartSeries ; ValueIndex : Longint ) : String
;
```

Description
This function returns the formatted Series.Mark text of the ValueIndexth Series point.

Marks Property

See also

Applies to

TChartSeries component

Declaration

```
property Marks : TSeriesMarks;
```

Description

The Marks Series subcomponent defines all necessary properties to draw a mark near to each Series point.

A mark consist of a colored rectangle with a text string on it and a line that indicates which points corresponds to which mark.

You can control all Marks formatting attributes and styles. The TChartSeries.OnGetMarkText event can be used to override the default Marks text strings or to hide specific point Marks.

Each different Series type draws it's marks differently.

See Also
TChartSeries.OnGetMarkText

MaxLabelsWidth Method

See also

Applies to

TChartAxis component

Declaration

```
function MaxLabelsWidth : Longint;
```

Description

This function returns the maximum width in Screen pixels of all Axis Labels.

It is internally used to calculate the Axis Increment in order to prevent overlapped Axis Labels.

See Also

LabelWidth

LabelHeight

MaxLegendWidth Method

See also

Applies to

TChartLegend component

Declaration

```
function MaxLegendWidth(NumLegendValues: LongInt): LongInt;
```

Description

The MaxLegendWidth function returns the width in pixels of the longest Legend item string.

The NumLegendValues parameter determines how many Legend items should be considered.

This function traverses all Series Titles or Series points (depending on Legend.LegendStyle) and calculates the maximum width.

It is used internally to calculate Legend's width.

See Also

TChartLegend.Font

TChartLegend.ColorWidth

TChartLegend.LegendStyle

MaxMarkWidth Method

See also

Applies to

TChart, TDBChart components

Declaration

```
function MaxMarkWidth : Longint;
```

Description

This function returns the Maximum width of the Active Series Maximum Marks witdh. Series Marks must be Visible.

That can be used to adjust the Chart Margins in order to accomodate the biggest Series Mark.

MaxMarkWidth Method

Applies to
TChartSeries component

Declaration
```
function MaxMarkWidth : LongInt;
```

Description
Returns the maximum width of all Series Marks.

See Also
SeriesMarks.Clip

MaxPointsPerPage Property

See also

Applies to
TChart component

Declaration
```
property MaxPointsPerPage : LongInt;
```

Description
Default Value: 0

The MaxPointsPerPage property controls "TeeChart AutoPaging".

Setting it to a number greater than zero makes TeeChart to internally divide Series points in Pages.

You can then navigate across Chart pages by using the Chart.Page and Chart.NumPages properties.

For each Page, TeeChart will automatically calculate and display the corresponding Series points.

The last page can have less number of points than other pages. You can use the Chart.ScaleLastPage to control if last page will be "stretched" or not.

See Also

TChart.ScaleLastPage

TChart.Page

TChart.NumPages

TChart.OnPageChange

MaxTextWidth Method

Applies to
TChart, TDBChart components

Declaration
```
function MaxTextWidth : LongInt;
```

Description
This function returns the Maximum width of the Active Series Maximum XLabels width. That applies only to Series which have XLabels.

MaxValue Property

See also

Applies to

TChartValueList component

Declaration

```
property MaxValue : Double;
```

Description

The MaxValue property returns the highest of all values inside the list.

As new points are being added to Series, the TChartValueList object calculates the MaxValue, MinValue and TotalABS properties.

This applies to all Series lists of values, such as XValues, YValues, etc.

Calling TChartValueList.RecalcMinMax method forces to recalculate MaxValue, MinValue and TotalABS values.

See Also

TChartValueList.MinValue

TChartValueList.TotalABS

TChartSeries.YValues

TChartValueList.RecalcMinMax

MaxXValue Example

```
Chart1.BottomAxis.Automatic := False ;
Chart1.BottomAxis.Maximum := Chart1.MaxXValue( Chart1.BottomAxis ) ;
```

MaxXValue Method

Example

Applies to

TChart, TDBChart components

Declaration

```
function MaxXValue (AAxis : TChartAxis): Double ;
```

Description

This function returns the calculted Maximum Horizontal value for the specified AAxis. AAxis can be TopAxis or BottomAxis.

Calculated means that the return value will the Maximum value of the Maximum Series X Values. Only Series with the HorizAxis equal to AAxis will be considered.

MaxXValue Method (TChartSeries)

Applies to
TChartSeries component

Declaration
```
function MaxXValue : Double; virtual;
```

Description
Returns the Maximum Value of the Series X Values List.

MaxYValue Example

```
Chart1.LeftAxis.Automatic := False ;
Chart1.LeftAxis.Maximum := Chart1.MaxYValue( Chart1.LeftAxis ) ;
```

MaxYValue Method

Example

Applies to

TChart, TDBChart components

Declaration

```
function MaxYValue (AAxis : TChartAxis ): Double;
```

Description

This function returns the calculted Maximum Vertical value for the specified AAxis. AAxis can be LeftAxis or RightAxis.

Calculated means that the return value will the Maximum value of the Maximum Series Y Values. Only Series with the VertAxis equal to AAxis will be considered.

MaxYValue Method (TChartSeries)

See also

Applies to

TChartSeries component

Declaration

```
function MaxYValue : Double; virtual;
```

Description

The MaxYValue function returns the highest of all the current Series Y point values.

Some special Series types override this function to calculate the maximum Y value correctly.

For example the TBubbleSeries component calculates the highest Y + Radius point.

See Also

TChartSeries.MinYValue

TChartSeries.MaxXValue

TChartSeries.MinXValue

TChart.MaxYValue

TChart.MinYValue

TChart.MaxXValue

TChart.MinXValue

MaxZOrder Property

See also

Applies to

TChart component

Declaration

```
property MaxZOrder : Longint
```

Description

Run-time and read only. The MaxZOrder property indicates the Chart depth in number of Series. When Chart.View3D is True, each Series has a Z order. The Series Z order defines the order along the Z axis when Series are displayed. The MaxZOrder property is the maximum Z order of all active Series. Some Series share a same Z order (stacked Bars, stacked Areas, etc).

See Also

TChartSeries.ZOrder

TChart.SeriesHeight3D

TChart.SeriesWidth3D

Maximum Property

See also

Applies to
TChartAxis component

Declaration
```
property Maximum:Double
```

Description
Axis Maximum is the highest value an Axis will use to scale their dependent Series point values.

It can be any number or DateTime value.

It must be greater than the Axis.Minimum value.

VERY IMPORTANT:

Axis.Automatic property must be FALSE.

If Axis.Automatic is True, the Axis will set Maximum and Minimum values to Maximum and Minimum dependent Series values.

See Also

Axis Minimum

Axis Increment

MinValue Property

See also

Applies to

TChartValueList component

Declaration

```
property MinValue : Double;
```

Description

Run-time and read only. The MinValue property returns the lowest of all values inside the list. See TChartValueList.MaxValue property for more information.

See Also

TChartValueList.MaxValue

TChartValueList.TotalABS

TChartSeries.YValues

TChartValueList.RecalcMinMax

MinXValue Example

```
Chart1.TopAxis.Automatic := False ;
Chart1.TopAxis.Minimum := Chart1.MinXValue( Chart1.TopAxis ) ;
```

MinXValue Method

Example

Applies to

TChart, TDBChart components

Declaration

```
function MinXValue (AAxis: TChartAxis): Double;
```

Description

This function returns the calculted Minimum Horizontal value for the specified AAxis. AAxis can be TopAxis or BottomAxis.

Calculated means that the return value will the Minimum value of the Minimum Series X Values. Only Series with the HorizAxis equal to AAxis will be considered.

MinXValue Method (TChartSeries)

Applies to
TChartSeries component

Declaration
```
function MinXValue : Double; virtual;
```

Description
Returns the Minimum Value of the Series X Values List.

MinYValue Example

```
Chart1.RightAxis.Automatic := False ;
Chart1.RightAxis.Minimum  := Chart1.MinYValue( Chart1.RightAxis ) ;
```

MinYValue Method

Example

Applies to

TChart, TDBChart components

Declaration

```
function MinYValue (AAxis: TChartAxis): Double;
```

Description

This function returns the calculted Minimum Vertical value for the specified AAxis. AAxis can be LeftAxis or RightAxis.

Calculated means that the return value will the Minimum value of the Minimum Series Y Values. Only Series with the VertAxis equal to AAxis will be considered.

MinYValue Method (TChartSeries)

Applies to
TChartSeries component

Declaration
```
function MinYValue : Double; virtual;
```

Description
Returns the Minimum Value of the Series Y Values Lists. As some Series have more than one Y Values List, this Minimum Value is the "Minimum of Minimums" of all Series Y Values lists.

Minimum Property

See also

Applies to

TChartAxis component

Declaration

```
property Minimum : Double;
```

Description

Axis Minimum is the lowest value an Axis will use to scale their dependent Series point values. Can be any number or DateTime value.

Must be lower than the Axis.Maximum value.

VERY IMPORTANT:

Axis.Automatic property must be FALSE. If Axis.Automatic is True, the Axis will set Maximum and Minimum values to Maximum and Minimum dependent Series values.

See Also

Axis Maximum

Axis

MinorTickCount Example

This code sets the number of Axis Minor ticks:

```
Chart1.LeftAxis.MinorTickCount := 4 ;
```

MinorTickCount Property

See also Example

Applies to

TChartAxis component

Declaration

```
property MinorTickCount : Integer
```

Description

Default 3

The MinorTickCount property determines the number of Axis minor ticks. Axis minor ticks are the Axis sub-ticks between major ticks.

It should be a positive number greater than zero and less than half the number of pixels between major ticks, otherwise Minor ticks will "overlap".

See Also

TChartAxis.MinorTickLength

TChartAxis.MinorTicks

MinorTickLength Property

See also

Applies to

TChartAxis component

Declaration

```
property MinorTickLength : Integer;
```

Description

Default 2

The MinorTickLength property indicates the length in pixels of Axis Minor ticks. You can control the number of Minor ticks by using the TChartAxis.MinorTickCount property. Minor ticks are displayed using the TChartAxis.MinorTicks pen property.

See Also

TChartAxis.MinorTickCount

TChartAxis.MinorTicks

MinorTicks Example

This code changes the Axis Minor tick pen:

```
With Chart1.BottomAxis do
begin
 MinorTickCount := 5 ;
 BottomAxis.MinorTickLength := 8 ; { pixels }
 BottomAxis.MinorTicks.Visible := True ;
 BottomAxis.MinorTicks.Color := clRed ;
 BottomAxis.MinorTicks.Width := 2 ;
end;
```

MinorTicks Property

See also Example

Applies to

TChartAxis component

Declaration

```
property MinorTicks : TChartPen;
```

Description

The MinorTicks property is the Pen used to draw the Axis Minor ticks. Minor ticks will only be displayed if MinorTicks.Visible is True.

See Also

TChartAxis.MinorTickCount

TChartAxis.MinorTickLength

Mode Property

Applies to
TChartPen component

Declaration
```
property Mode : TPenMode;
```

Description
The Mode property determines how the pen draws lines on the canvas.

See Delphi help: TPen.Mode

Monochrome Property

See also

Applies to

TChart component

Declaration

```
property Monochrome : Boolean;
```

Description

Default Value: False

The Monochrome property will allow Charts to be draw using only black and white colors. It's not yet completely implemented. Right now it uses the internal Bitmap.Monochrome property, and the output quality isn't really good.

See Also
TChart.MonochromePrinting

MonochromePrinting Property

See also

Applies to
TChart component

Declaration
```
property MonochromePrinting : Boolean:
```

Description
Default Value: True

The MonochromePrinting property will allow Charts to be sent to the printer in black and white or color mode. This property will NOT change any Chart color. It will ONLY set the Printer to use the Black & White printer's mode. (If the printer is a color printer, like a HP850C printer).

The Chart.Monochrome property will allow charts to be drawn using Black & White colors only.

See Also
TChart.Monochrome

MultiArea Property

See also

Applies to

TAreaSeries component

Declaration

```
property MultiArea : TMultiArea
```

Description

Default maNone

The MultiArea property determines the kind of displayed Area when there's more than one AreaSeries with the same ParentChart. The default value is maNone, meaning all Areas will be drawn one behind the other. maStacked and maStacked100 modes will draw each Area on top of previous one. msStacked100 adjusts each individual point to a common 0..100 axis scale. The order which Series are accumulated depends on the Chart.SeriesList property.

See Also

TChartSeries.PointOrigin

MultiBar

MultiBar Property

See also

Applies to

TBarSeries and THorizBarSeries component

Declaration

```
property MultiBar : TMultiBar
```

Description

If you have more than one TBarSeries in the same Chart, then you can choose if they will be drawn side-by-side, back-to-front or Stacked. Side-by-side means the Bar width will be divided by the number of Bar Series.

See Also

TChartSeries.PointOrigin

TAreaSeries.MultiArea

Multiplier Property

Example

Applies to

TChartValueList component

Declaration

```
property Multiplier : Double;
```

Description

The Multiplier property will be used as a factor to convert all Series points X and / or Y values.

Multiplier property Example

```
LineSeries1.YValues.Multiplier := -5 ;
```

Name Property

Applies to
TChartValueList component

Declaration
```
property Name : String;
```

Description
The Name property is used to identify all Series lists of values. All Series have an "Y" and "X" value lists. Some Series have more lists of values, such as TCandleSeries, which have OpenValues, CloseValues, HighValues and LowValues lists.

NextPage Method

See also

Applies to

TChart, TDBChart components

Declaration

```
procedure NextPage;
```

Description

When MaxPointsPerPage is greater than Zero, TeeChart automatically divides point values in Pages. Calling NextPage is the same as Page := Page + 1 The NumPages chart property returns the total number of pages.

See Also

MaxPointsPerPage

ScaleLastPage property

PreviousPage

NextTask Property

See also Example

Applies to

TGanttSeries component

Declaration

```
property NextTask : TChartValueList;
```

Description

The NextTask property is a TList component that holds the Gantt bar index each Gantt bar is connected to.

When a Gantt bar is added to TGanttSeries, it's NextTask value is assigned to -1 by default. That means the Gantt bar is NOT connected to any other Gantt Bar.

You need to set a valid bar index to NextTask.

NextTask property Example

Example:

Let's add two Gantt bars:

```
GanttSeries1.Clear;
GanttSeries1.AddGantt( EncodeDate(1997,1,1),
            EncodeDate( 1997,1,31),
            5, { <-- vertical user defined position }
            'Some period...' );
GanttSeries1.AddGantt( EncodeDate(1997,3,1),
            EncodeDate( 1997,3,31),
            6, { <-- vertical user defined position }
            'Other period...' );
```

Now let's "connect" the first Gantt bar to the second:

```
GanttSeries1.NextTask[ 0 ] := 1 ;
GanttSeries1.Repaint;
```

See Also
TGanttSeries.AddGantt

NormalBarColor Property

See also

Applies to
TCustomBarSeries component

Declaration
```
property NormalBarColor : TColor;
```

Description
Run-time and read only.

The NormalBarColor property returns the color used to fill the Bar sides.

Each different Bar style uses NormalBarColor to fill a different Bar area.

See Also

TCustomBarSeries.DarkColor

TCustomBarSeries.DarkerColor

NumPages Method

See also

Applies to

TChart component

Declaration

```
function NumPages : Longint;
```

Description

The NumPages function returns the number of Chart pages. The TChart.MaxPointsPerPage property must be greater than zero to activate auto paging.

The TChart.Page property determines the current visible page.

This code traverses all Chart pages:

```
for t := 1 to Chart1.NumPages do Chart1.Page := t ;
```

See Also

TChart.OnPageChange

TChart.MaxPointsPerPage

TChart.Page

NumSampleValues Example

The NumSampleValues function is now a "Class" function and can be called directly without creating any instance:

```
Var tmp : Longint;
tmp := TLineSeries.NumSampleValues;
```

instead of:

```
With TLineSeries.Create(Self) do
try
 tmp := NumSampleValues;
finally
 Free;
end;
```

NumSampleValues Method

See also Example

Applies to

TChartSeries component

Declaration

```
function NumSampleValues : Longint;
```

Description

Each Series draws random values at design mode unless you connect the Series component to other Series component or to a DataSet (Table,SQL,TClientDataset) component.

This virtual function returns the predefined number of random values each Series draws.

See Also
FillSampleValues method

OffsetPercent Example

This code sets 3 Bar series OffsetPercent to achieve an overlayed Bar chart:

```
BarSeries1.OffsetPercent := 25 ; { % of Bar width }
BarSeries2.OffsetPercent := 50 ; { % of Bar width }
BarSeries3.OffsetPercent := 75 ; { % of Bar width }
```

OffsetPercent Property

See also Example

Applies to

TBarSeries and THorizBarSeries components (TCustomBarSeries)

Declaration

```
property OffsetPercent : Integer
```

Description

Default 0

The OffsetPercent property indicates the Bar displacement in percent of Bar size. Displacement is horizontal for TBarSeries and vertical for THorizBarSeries. This property can be used to create "overlayed" Bar charts. You can specify positive or negative values.

See Also

TBarSeries.BarWidth

THorizBarSeries.BarHeight

TCustomBarSeries.SideMargins

TCustomBarSeries.BarWidthPercent

OnAfterAdd Example

Here we use the OnAfterAdd event to arrange the XAxis scale as new points are added. }

```
procedure TScrollForm.LineSeries1AfterAdd(Sender: TChartSeries;
 ValueIndex: Longint);
begin
 With Sender.GetHorizAxis do { <-- with the Horizontal Axis... }
  Begin
   Automatic := False;  { <-- we dont want automatic scaling }
   { In this example, we will set the Axis Minimum and Maximum values to
  show One Hour of data ending at last point Time plus 5 minutes}
   Minimum := 0;
   Maximum := Sender.XValues.MaxValue +
            DateTimeStep[dtFiveMinutes ];
   Minimum := Maximum - DateTimeStep[ dtOneHour ];
  end;
end;
```

OnAfterAdd Event

Example

Applies to

TChartSeries component

Declaration

```
property OnAfterAdd : TSeriesOnAfterAdd;
```

Description

Every Series component has an OnAfterAdd event. This event is triggered whenever a new point has been Added or Inserted.

OnAfterDraw Event

See also

Applies to
TChart component

Declaration
```
property OnAfterDraw: TNotifyEvent;
```

Description
This event gets called just after TChart has drawn all series.

Warning:
Do not modify any property in this event that would force a TChart repaint. Doing this will cause a recursive loop.

See also

TChartSeries.AfterDrawValues

TChartSeries.BeforeDrawValues

OnAllowScroll Event

Applies to
TChart component

Declaration
```
property OnAllowScroll: TChartAllowScrollEvent;
```

Description

Warning:
TChartAllowScrollEvent=Procedure (Sender:TChartAxis;
Var Amin, AMax: Double;
Var AllowScroll:Boolean) of object;

OnBeforeAdd Event

Example

Applies to

TChartSeries component

Declaration

```
property OnBeforeAdd : TSeriesOnBeforeAdd;
```

Description

Every Series component has an OnBeforeAdd event. This event is triggered whenever a new point is going to be Added or Inserted. You can delegate this event in your Form's code to, for example, control if new points will be really added to the Series. This event also happens when a Series component is connected to a Table or Query.

OnBeforeAdd Example

Example 1 : In this example we'll deny points with Y values less than 100 to be added:

```
Function TForm1.OnBeforeAdd ( Sender : TChartSeries ) : Boolean ;
Begin
  result := Sender.YValues.TempValue >= 100 ;
End ;
```

Example 2 : In this example we'll filter which Table records are valid : (Assuming Table1 is the Series.DataSource)

```
Function TForm1.OnBeforeAdd ( Sender : TChartSeries ) : Boolean ;
Begin
  result := Table1MyDateField.AsDateTime >= StrToDate ( '1/1/96' ) ;
End ;
```

OnClearValues Event

Applies to
TChartSeries component

Declaration
```
property OnClearValues : TSeriesOnClear;
```

Description
This event will notify you when the Series Clears and removes their Point values.

OnClick Event

See also Example

Applies to

TChartSeries component

Declaration

```
property OnClick : TSeriesClick
```

Description

TChartSeries components will notify you when users click on a Series point. Each TChartSeries component calculates if mouse coordinates are over a point screen regions. For example, TPieSeries point regions are the Pie Sectors. TBarSeries detects clicked Bar regions. The ValueIndex parameter is the Series point number.

See Also
TChartSeries.OnDblClick

OnClick Example

We will show a message to the user when he or she clicks a PieSeries chart:

```
procedure TForm1PieSeriesOnClick( Sender:TChartSeries; ValueIndex:Longint;
  Button:TMouseButton; Shift: TShiftState; X, Y: Integer);
Begin
ShowMessage ( 'This point value: ' + FloatToStr ( Sender.YValues
  [ ValueIndex ] ) ) ;
End;
```

OnClickAxis Event

See also

Applies to
TChart component

Declaration
```
property OnClickAxis : TChartClickAxis;
```

Description
An OnClickAxis event occurs whenever the user clicks near to a Chart axis subcomponent.

The Sender parameter specifies the Chart component that triggered the event.

The Axis parameter is the corresponding clicked Chart axis. It can be the LeftAxis, RightAxis, TopAxis or BottomAxis Chart subcomponent.

The Button, Shift, X and Y parameters determine the mouse button and mouse cursor coordinates at the time the axis was clicked.

This event has a higher priority than OnClickBackGround event and lower priority than OnClickSeries event.

See Also

TChart.OnClickBackGround

TChart.OnClickLegend

TChart.OnClickSeries

TChart.OnDblClick

OnClickBackground Event

See also

Applies to

TChart component

Declaration

```
property OnClickBackground : TChartClick;
```

Description

An OnClickBackGround event occurs whenever the user clicks onto a Chart space outside Axis, Series points or Chart Legend.

It has the lower priority of all other mouse events.

The Sender parameter specifies the Chart component that originated the event.

The Button, Shift, X and Y parameters determine the mouse button and mouse cursor coordinates at the time the Chart was clicked.

See Also

TChart.OnClickAxis

TChart.OnClickLegend

TChart.OnClickSeries

TChart.OnDblClick

OnClickLegend Event

See also

Applies to

TChart component

Declaration

```
property OnClickLegend : TChartClick;
```

Description

An OnClickLegend event occurs whenever the user clicks onto Chart Legend rectangle.

The Sender parameter specifies the Chart component that originated the event.

The Button, Shift, X and Y parameters determine the mouse button and mouse cursor coordinates at the time the Chart Legend was clicked.

See Also

TChart.OnClickAxis

TChart.OnClickBackGround

TChart.OnClickSeries

TChart.OnDblClick

OnClickPointer Event

Applies to
TChartSeries component

Declaration
```
property OnClickPointer : TSeriesClickPointerEvent
```

Description
TSeriesClickPointerEvent=Procedure(Sender:TCustomSeries; ValueIndex:Longint; X, Y: Integer) of object; This event is similar to the OnClick event except it will only trigger when mouse clicks over Pointers, not Lines.

OnClickSeries Example

This event procedure shows the user at which series point (any drawn part of series) they have clicked.

```
procedure TForm1.DBChart1ClickSeries (Sender: TCustomChart;
 Series: TChartSeries; ValueIndex: Longint; Button: TMouseButton;
 Shift: TShiftState; X, Y: Integer);
begin
 ShowMessage(' Clicked Series: '+Series.Name+' at point: '+
  inttostr(valueindex));
end;
```

OnClickSeries Event

See also Example

Applies to

TChart component

Declaration

```
property OnClickSeries : TChartClickSeries;
```

Description

An OnClickSeries event occurs whenever the user clicks onto any Chart Series point. The Sender parameter specifies the Chart component that originated the event. The Series parameter is the corresponding clicked Series component, and the ValueIndex parameter refers to the exact clicked point in the Series. Series have also an OnClick event that can be used individually to catch clicked points. The Button, Shift, X and Y parameters determine the mouse button and mouse cursor coordinates at the time the Series point was clicked.

WARNING:

Use the CancelMouse property to control how the mouse button behaves with dual modes (scroll or zoom after a ClickOnSeries event)

See Also

TChart.OnClickAxis

TChart.OnClickBackGround

TChart.OnClickLegend

TChart.OnDblClick

TChartSeries.OnClick

TChartSeries.OnClickPointer

OnDblClick Event

See also

Applies to
TChart component

Declaration
```
property OnDblClick : TNotifyEvent;
```

Description
An OnDblClick event occurs whenever the user double clicks onto a Chart. The Sender parameter specifies the Chart component that originated the event.

You must cast the Sender parameter to TChart or TDBChart class to use its properties and methods:

```
(Sender as TChart).Color := clYellow ;
```

To retrieve if user double clicked onto a Series point, you can use this code:

```
procedure TForm1.Chart1DblClick(Sender: TObject);
var tmp:Longint;
begin
tmp:=BarSeries1.GetCursorValueIndex;
if tmp<>-1 then
  begin
{ show the point index }
  ShowMessage('You double-clicked the: '+IntToStr( tmp )+' point !');
{ tell TeeChart NOT to continue with zoom or scroll }
  CancelMouse;
  end;
end;
```

The TChartSeries.GetCursorValueIndex function returns the corresponding point under mouse cursor position. It returns -1 if no point is under mouse cursor. The TChartSeries.GetCursorValues function returns the mouse cursor X and Y position in Axis scales coordinates. The TChartSeries.XScreenToValue and TChartSeries.YScreenToValue functions convert from screen pixel coordinates to Series scales.

WARNING:
Use the CancelMouse property to control how the mouse button behaves with dual modes (scroll or zoom after the OnDblClick event)

Example:

```
ShowMessage('Double-Clicked Chart!');
CancelMouse;
```

See Also

TChartSeries.GetCursorValueIndex

TChartSeries.GetCursorValues

TChartSeries.XScreenToValue

TChartSeries.YScreenToValue

TChartSeries.CalcXPos

TChartSeries.CalcYPos

TChartSeries.CalcXPosValue

TChartSeries.CalcYPosValue

OnDblClick Event (TchartSeries)

See also

Applies to

TChartSeries component

Declaration

```
property OnDblClick : TSeriesClick
```

Description

TChartSeries components will notify you when users double click on a Series point. This is similar to the TChartSeries.OnClick event Each TChartSeries component calculates if mouse coordinates are over a point screen regions. For example, TPieSeries point regions are the Pie Sectors. TBarSeries detects clicked Bar regions. The ValueIndex parameter is the Series point number.

See Also
TChartSeries.OnClick

OnGetAxisLabel Event

Applies to
TChart, TDBChart components

Declaration
```
property OnGetAxisLabel : TAxisOnGetLabel
```

Description
An Event is triggered for each Axis Label painted. There are two different uses for OnGetAxisLabel:

1) : Axis Labels are Values. Is this case, the Series parameter will be nil, and the ValueIndex will be -1.

2) : Axis Labels are Series points. The Series parameter will be a valid TChartSeries, and the ValueIndex will be the current Series point position. You can change the LabelText referred parameter for drawing a different Axis Label.

OnGetBarStyle Event

See also Example

Applies to

TCustomBarSeries components

Declaration

```
property OnGetBarStyle : TGetBarStyleEvent;
```

Description

The OnGetBarStyle event is called for every Bar that's going to be displayed. You can specify a different Bar style for each Bar point. Using this event overrides the BarStyle property.

OnGetBarStyle Example

This code will show specific Bar styles depending on Bar's Y values:

```
procedure TForm1.BarSeries1GetBarStyle(Sender: TCustomBarSeries; ValueIndex:
  Longint; var TheBarStyle: TBarStyle);
begin
  if Sender.YValue[ ValueIndex ] < 1000 then
  TheBarStyle := bsPyramid
  else
    TheBarStyle := bsRectangle ;
end;
```

See Also
TCustomBarSeries.BarStyle

OnGetLegendPos Event

See also

Applies to

TChart component

Declaration

```
property OnGetLegendPos : TOnGetLegendPos;
```

Description

An OnGetLegendPos event occurs whenever a Legend item is going to be displayed.

You can used this event to force specific X , Y and / or XColor Legend pixel coordinates.

The Chart component calculates the default X,Y and XColor parameters before calling this event.

The Sender parameter is the Chart component that originated the event.

The Index parameter corresponds to the specific Legend item number going to be displayed. It can be either an Active Series Title or a Series point index, depending on the TChartLegend.LegendStyle property.

See Also

TChartLegend.LegendStyle

TChart.OnGetLegendRect

OnGetLegendRect Event

See also

Applies to

TChart component

Declaration

```
property OnGetLegendRect : TOnGetLegendRect;
```

Description

An OnGetLegendRect event occurs whenever the Chart Legend is going to be displayed.

You can use this event to force an specific Legend position and dimensions.

The Rect parameter determines the default Legend position and dimensions in screen pixels.

After drawing the Legend, the available Chart space will be decreased to not overlap with Legend unless you set the TChartLegend.ResizeChart property to False.

When using this event you'll need to use the TChart.OnGetLegendPos event as well, to supply the exact coordinates for each Legend item.

See Also

TChartLegend.Alignment

TChartLegend.OnGetLegendPos

OnGetLegendText Event

Example

Applies to

TChart, TDBChart components

Declaration

```
property OnGetLegendText : TOnGetLegendText
```

Description

You can use this event to override the text strings that TChartLegend paints.

TChartLegend has two different modes (or LegendStyles):

lsSeries	Legend contains Series Names or Titles.
lsValues	Legend contains a Series values or labels.

If the LegendStyle is lsAuto, TChartLegend paints in lsSeries mode if more than one Active Series exists in the Chart.

OnGetLegendText Example

By using this event, you can set the LegendText string parameter to your customized text:

```
procedure TForm1.DBChart1GetLegendText(Sender: TCustomChart; LegendStyle:
  TLegendStyle; Index: Longint; var LegendText: OpenString);
begin
if LegendStyle = lsSeries then
  LegendText := 'My Series #: ' + IntToStr( Index )
else
  LegendText := 'Value #: '+ IntToStr( Index ) ;
end;
```

OnGetMarkText Event

Example

Applies to

TChartSeries component

Declaration

```
property OnGetMarkText : TSeriesOnGetMarkText;
```

Description

This event notifies the user that a Mark Text String must be supplied. MarkText string contains the default text representation. You can alter MarkText for the Series to paint a customized mark.

OnGetMarkText Example

This example will show a customized point Mark at Point Index: 3

```
Procedure TForm1LineSeries1GetMarkText( Sender : TChartSeries ; ValueIndex :
  Longint ; Var MarkText : String )
Begin
  if ValueIndex = 3 then
    MarkText := 'Hello World';
End;
```

OnGetNextAxisLabel Event

See also

Applies to

TChart component

Declaration

```
property OnGetNextAxisLabel : TAxisOnGetNextLabel;
```

Description

An OnGetNextAxisLabel event is used to define custom Axis Labels. Using this event you can customize Axis Labels positions and values.

This event gets called in a loop until you set the Stop parameter to False OR the LabelValue parameter is BIGGER than the Axis Maximum value.

The Stop parameter is True by default, meaning that if it's not set to False the first time this event gets called, TeeChart will draw the default Axis Labels.

The Sender parameter specifies the Axis subcomponent. It can be the Chart LeftAxis, RightAxis, TopAxis or BottomAxis axis components.

The LabelIndex parameter is an incremental counter for you to know which Label the event is asking for a value.

The LabelValue parameter must be filled with the desired Axis Label value.

You can use the TChart.OnGetAxisLabel event to override the default Axis Labels text with your preferred Axis Label string representation.

An example of use of this event can be as follows:

```
procedure TAxisLabelsForm.Chart1GetNextAxisLabel(Sender: TChartAxis;
  LabelIndex: Longint; var LabelValue: Double; var Stop: Boolean);
 begin
  if Sender=Chart1.LeftAxis then
  Begin
   { In this example, we want the Vertical Left Axis to show
    labels only for positive values, starting at zero and
    with 250 label increment.
   }
   if LabelValue>=250 then LabelValue:=LabelValue+250
              else LabelValue:=250;
  End;
  { we want more labels !! }
  Stop:=False;
 end;
```

See Also

TChart.OnGetAxisLabel

TChartAxis.Increment

TChartAxis.ExactDateTime

OnPageChange Event

See also Example

Applies to

TChart component

Declaration

```
property OnPageChange : TNotifyEvent;
```

Description

An OnPageChange event occurs whenever the TChart.Page property has been changed and before the Chart repaints the new page points.

The TChart.MaxPointsPerPage property must be greater than zero to activate the automatic paging mechanism.

This event DOES NOT get called the first time the TChart.Page property is set to 1.

OnPageChange event Example

You can use this event to, for example, display the current Chart page:

```
procedure TForm1.Chart1PageChange(Sender: TObject);
begin
  Label1.Caption:='Current page: '+IntToStr( (Sender as TChart).Page )+ ' of
  '+IntToStr((Sender as TChart).NumPages ) ;
end;
```

See Also

TChart.MaxPointsPerPage

TChart.Page

TChart.PreviousPage

TChart.NextPage

TChart.NumPages

OnProcessRecord Event

See also

Applies to
TDBChart component

Declaration
```
property OnProcessRecord : TProcessRecordEvent;
```

Description
An OnProcessRecord event occurs for each record that is retrieved from the database.

The TDBChart component triggers this event whenever a new point has been added to the Series.

The DataSet parameter is the database component (Table, Query, etc) from which records are being loaded.

You can raise an Abort silent exception in this event to notify TDBChart to stop retrieving records and to NOT add the current record point.

See Also

TDBChart.RefreshData

TDBChart.RefreshDataSet

TChartSeries.DataSource

OnScroll Event

See also

Applies to

TChart component

Declaration

```
property OnScroll : TNotifyEvent;
```

Description

An OnScroll event occurs when users scroll Chart contents by dragging the right mouse button.

This event gets called repeteadly while users move the mouse BEFORE the Chart component is repainted to show the new Axis scales.

You can use this event toghether with TChart.OnZoom and TChart.OnUndoZoom events to know changes on Axis scales.

See Also

TChartAxis.Scroll

TChartAxis.SetMinMax

TChart.OnUndoZoom

TChart.OnZoom

OnUndoZoom Event

See also Example

Applies to

TChart component

Declaration

```
property OnUndoZoom : TNotifyEvent;
```

Description

An OnUndoZoom event occurs when Chart Axis scales are reset to the Minimum and Maximum values that fit all Series points. Users can undo zoom by dragging the left mouse button drawing a rectangle from bottom-right to top-left. You can undo zoom by calling the TChart.UndoZoom procedure. Both techniques cause this event to be triggered. You can use this event together with TChart.OnZoom and TChart.OnScroll events to react to changes in Axis scales.

OnUndoZoom Example

This code will reset axis scales at UndoZoom event:

```
procedure TForm1.Chart1UndoZoom(Sender: TObject);
begin
  With Sender as TChart do
  begin
    LeftAxis.SetMinMax( 0, 100 );
    BottomAxis.SetMinMax( EncodeDate( 1996,1,1), EncodeDate( 1996,12,31) );
  end;
end;
```

See Also

TChart.OnScroll

TChart.UndoZoom

TChart.OnZoom

TChart.ZoomPercent

TChart.ZoomRect

TChartAxis.Scroll

TChartAxis.SetMinMax

OnZoom Event

See also

Applies to

TChart component

Declaration

```
property OnZoom : TNotifyEvent;
```

Description

An OnZoom event occurs whenever Chart contents is being zoomed.

Zoom can be performed by dragging the mouse at run-time or by calling the TChart.ZoomRect or TChart.ZoomPercent methods.

This event gets called BEFORE the Chart component is repainted to show the new Axis scales.

You can use this event toghether with TChart.OnUndoZoom and TChart.OnScroll events to know changes on Axis scales.

See Also

TChart.OnScroll

TChart.OnUndoZoom

TChart.UndoZoom

TChart.ZoomPercent

TChart.ZoomRect

TChartAxis.Scroll

TChartAxis.SetMinMax

Order Example

This code creates a new TLineSeries component that draws a triangle:

```
LineSeries1.Clear;
LineSeries1.XValues.Order:=loNone;
LineSeries1.AddXY(-100, 0,'',clRed);
LineSeries1.AddXY(  0,100,'',clBlue);
LineSeries1.AddXY( 100, 0,'',clGreen);
LineSeries1.AddXY(-100, 0,'',clYellow); { the closing line }
```

Order Property

See also Example

Applies to

TChartValueList component

Declaration

```
property Order : TChartListOrder
```

Description

Run-time only. The Order property determines if points will be automatically sorted or it they will remain always at their original positions. This Order is used by default by the Series XValues to draw lines from Left to Right. Setting the XValues.Order property to loNone will respect the points order at point creation. This can be used to draw polygons.

See Also
TChartValueList

OriginalCursor Property

Unit

Chart

Applies to

All TeeChart components

Declaration

```
property OriginalCursor : TCursor;
```

Description

This public property returns the original Chart.Cursor property. Used to reset the cursor when the mouse moves over the series.

OtherSide Property

See also

Applies to

TChartAxis component

Declaration

```
property OtherSide : Boolean;
```

Description

Run-time and read only. This property returns true if the Axis is the RightAxis or TopAxis. Chart.TopAxis and Chart.RightAxis are in the OtherSide. Chart.LeftAxis and Chart.BottomAxis are not in the OtherSide.

Chart Axis	Horizontal	OtherSide
LeftAxis	False	False
TopAxis	True	True
RightAxis	False	True
BottomAxis	True	False

See Also

Axis.Horizontal function

TChartAxis.Horizontal

TChartAxis

Owner Property

Applies to
TChartValueList component

Declaration
```
property Owner : TChartSeries;
```

Description
Read-only and run time.

The Owner property returns the TChartSeries that owns the TChartValueList.

It can be only accessed at run-time and it's read-only.

Page Property

See also

Applies to

TChart component

Declaration

```
property Page : LongInt;
```

Description

Run-time only.

The Page property determines the current visible points of Series in a Chart with MaxPointsPerPage property greater than zero.

When TChart.MaxPointsPerPage property is greater than zero, TeeChart internally divides all Series points in "pages".

Each page is assigned a different initial and ending X coordinates.

The TChart.NumPages property returns the total number of pages. It equals to the total number of Series points divided the MaxPointsPerPage property.

The TChart.PreviousPage and TChart.NextPage methods decrement or increment the Page property respectively.

See also

TChart.MaxPointsPerPage

TChart.NextPage

TChart.NumPages

TChart.PreviousPage

ParentChart Example

In Form1, we'll create and show another Form (Form2), and assign Form1.LineSeries1 to Form2.ChartInForm2 :

```
With TForm2.Create(Self) do
try
  Self.LineSeries1.ParentChart := ChartInForm2 ;
  ShowModal ;
finally
  Free ;
end ;
```

That will show Form2 (containing a Chart component) and drawing Form1.LineSeries1.

ParentChart Property

Example

Applies to

TChartSeries component

Declaration

```
property ParentChart : TCustomChart
```

Description

The Series.ParentChart propery is mandatory. Each Series component must be "attached" to either a TChart or a TDBChart component. You can change which Chart component will "own" the Series both at design and run-time. When you add a series to a Chart using the Chart editor, ParentChart is set automatically - otherwise you may set it via the Object Inspector.

Advanced:

You can also set the Series.ParentChart property to Chart components in different Forms:

ParentChart Property (TChartTitle)

Applies to
TChartTitle component

Declaration
```
property ParentChart : TChart;
```

Description
The ParentChart property is a reference to the TChart component that owns the TChartTitle.

ParentSeries Property (TSeriesPointer)

See also

Applies to
TSeriesPointer component

Declaration
```
property ParentSeries : TChartSeries;
```

Description
Run-time and read only.

The ParentSeries property returns the TChartSeries owner of Pointer subcomponent.

ParentSeries Property (TSeriesMarks)

See also

Applies to
TSeriesMarks component

Declaration
```
property ParentSeries : TChartSeries;
```

Description
Read-only and run time.

The ParentSeries property returns the Series component that owns the TSeriesMarks subcomponent.

All Series types own a Marks subcomponent of TSeriesMarks class.

See Also
TChartSeries.Marks

See Also
TPointSeries.Pointer

PatternColors Property

See also

Applies to

TChartLegend component

Declaration

```
property PatternColors : Boolean;
```

Description

Default True

Run-time only. The PatternColors property indicates Legend will use the corresponding Brush style of each Series to fill the Legend.ColorWidth rectangle. It's available at run-time. It will probably be included at design-mode in future TeeChart upgrades. Some Series like TPieSeries allow to specify a different Brush style for each Point.

See Also

TChartLegend.ColorWidth

TLineSeries.LineBrush

TAreaSeries.AreaBrush

TPieSeries.UsePatterns

TBarSeries.BarBrush

PatternPalette Global Constant

See also

Applies to
Global Constant

Declaration

```
Const MaxDefaultPatterns = 6;
   PatternPalette : Array[1..MaxDefaultPatterns] of TBrushStyle =
   ( bsHorizontal, bsVertical, bsFDiagonal, bsBDiagonal, bsCross,
bsDiagCross);
```

Description
The PatternPalette stores a default set of Brush patterns. These patterns are used in TPieSeries component to paint each different pie slice with a different brush style.

A future TeeChart version will increment the number of available brush patterns.

See Also
TPieSeries.UsePatterns

Pen Property (TChartWall)

See also

Applies to

TChartWall component

Declaration

```
property Pen : TChartPen
```

Description

The Pen property determines the kind of pen used to draw the Chart Walls frame. The Chart.View3DWalls property should be True to make walls visible.

Pen Property

See also

Applies to

TSeriesPointer component

Declaration

```
property Pen : TChartPen;
```

Description

The Pen property determines the kind of pen used to draw a frame around Series Pointers.

You can make it invisible by setting Pen.Visible property to False.

See Also

TChart.View3DWalls

TChartWall.Brush

TChartWall.Color

See Also

TSeriesPointer.Brush

TSeriesPointer.Visible

PercentFormat Example

```
BubbleSeries1.PercentFormat := '##0.0# %';
```

See Delphi help under FormatFloat Function for complete details.

PercentFormat Property

Example

Applies to

TChartSeries component

Declaration

```
property PercentFormat : String;
```

Description

Chart Series components have a PercentFormat property. PercentFormat is a standard Delphi formatting string specifier. It is used to draw the Series Marks Percent Style figures.

PercentOf Global Constant

Example

Applies to
All TeeChart components

Declaration
```
property PercentOf : String[10];
```

Description
The PercentOf global string variable contains the default text separator between percents and totals.

The default value is " of ".

This shows: " 23 % of 34555 "

Changing PercentOf at runtime will NOT automatically redraw the Chart. You will need to call Chart.Invalidate for the Chart to being repainted.

PercentOf Global Constant Example

```
PercentOf := ' of total: ' ;
Chart1.Invalidate ;
```

PiePen Property

See also

Applies to

TCircledSeries component

Declaration

```
property PiePen : TChartPen;
```

Description

The PiePen property determines the kind of pen used to draw the outmost circle. TPolarSeries uses CirclePen.

See Also
TCircledSeries.Circled

PieValues Property

See also

Applies to
TPieSeries component

Declaration
```
property PieValues : TChartValueList;
```

Description
The PieValues property stores the Pie slice values. It's a standard TList, so you can access it as follows:
```
tmp := PieSeries1.PieValues[ 3 ] ;
```

Warning:
You should call AddPie to ADD Pie points and Delete to REMOVE Pie points.

See Also

TPieSeries.AddPie

TChartSeries.Delete

PointOrigin Method

See also

Applies to

TChartSeries component

Declaration

```
function PointOrigin(ValueIndex: Longint; SumAll: Boolean): Double;
```

Description

The PointOrigin function returns the summed values of more than one Series point. The summed values are those returned by GetMarkValue function. It's only used by Series types with Stacked or Stacked 100% styles such as TBarSeries, THorizBarSeries and TAreaSeries.

See Also

TChartSeries.GetMarkValue

TBarSeries.MultiBar

THorizBarSeries.MultiBar

TAreaSeries.MultiArea

PointToAngle Method

See also

Applies to

TCircledSeries component

Declaration

```
function PointToAngle( x , y : Longint ) : Double;
```

Description

The PointToAngle function returns the angle from the XY point parameter to the circle center. It currently works with Circled ellipses.

The formula used to calculate the Angle is:

```
result:=ArcTan(Abs(y-CircleYCenter)/Abs(x-CircleXCenter));
```

Angles are returned in radians from 0 to 2*PI.

PointToAngle Method (TPieSeries)

Applies to
TPieSeries component

Declaration
```
function PointToAngle ( x , y : Longint ) : Double ;
```

Description
Given a point in pixel coordinates (x,y) this function returns the corresponding Angle (0 to 360). This is used internally by TPieSeries to calculated the Clicked Pie Sector.

See Also

TCircledSeries.AngleToPos

TCircledSeries.RotationAngle

TCircledSeries.XRadius

TCircledSeries.YRadius

Pointer Property

Applies to
TPointSeries component

Declaration
```
property Pointer : TSeriesPointer;
```

Description
The Pointer property is a subcomponent of TPointSeries, TLineSeries and all other derived TPointSeries components like TBubbleSeries.

Each point in a TPointSeries is drawn using the Pointer properties.

Pointer contains several properties to control the formatting attributes of Points like Pen, Brush, Draw3D, Visible, etc.

Please refer to TSeriesPointer help page for a complete description of TSeriesPointer subcomponent.

PositionPercent Property

See also

Applies to
TChartAxis component

Declaration
```
property PositionPercent : Integer;
```

Description
Default = 0

The PositionPercent property defines the position, as a percentage of Chart width or height (depending on whether it is applied to a Horizontal Axis or a Vertical Axis) of the Position of the Axis to which it is applied. Left and Top are 0,0 positions.

Use with StartPosition and EndPosition properties to define the Axis position on a Chart. You may use these properties with default and additional Axis, see the example below:

Example
```
//Creates a new Vertical Axis and defines a position for it.
procedure TForm1.BitBtn2Click(Sender: TObject);
Var MyAxis : TChartAxis ;
begin
 MyAxis := TChartAxis.Create( Chart1 );
 Series2.CustomVertAxis := MyAxis;
 //You can modify any property of the new created axes, such as the axis
  color or axis title
 With MyAxis do
 begin
  Axis.Color:=clGreen ;
  Title.Caption := 'Extra axis' ;
  Title.Font.Style:=[fsBold];
  Title.Angle := 90;
  PositionPercent := 20; //percentage of Chart rectangle
  StartPosition:=50;
  EndPosition:=100;
 end;
end;
```

See Also

TChartAxis.StartPosition

TChartAxis.EndPosition

TChartAxis.CustomDraw

TChartAxis.CustomDrawMinMax

EndPosition Property

See also

Applies to

TChartAxis component

Declaration

```
property EndPosition : Integer;
```

Description

Default = 0

The EndPosition property defines the position, as a percentage of Chart width or height (depending on whether it is applied to a Horizontal Axis or a Vertical Axis) of the End position of the Axis to which it is applied. Left and Top are 0,0 positions.

Use with StartPosition and PositionPercent properties to define the Axis position on a Chart.

See Also

TChartAxis.StartPosition

TChartAxis.PositionPercent

TChartAxis.CustomDraw

TChartAxis.CustomDrawMinMax

StartPosition Property

See also

Applies to

TChartAxis component

Declaration

```
property StartPosition : Integer;
```

Description

Default = 0

The StartPosition property defines the position, as a percentage of Chart width or height (depending on whether it is applied to a Horizontal Axis or a Vertical Axis) of the Start position of the Axis to which it is applied. Left and Top are 0,0 positions.

Use with PositionPercent and EndPosition properties to define the Axis position on a Chart.

See Also

TChartAxis.EndPosition

TChartAxis.PositionPercent

TChartAxis.CustomDraw

TChartAxis.CustomDrawMinMax

PosAxis Property

See also Example

Applies to

TChartAxis component

Declaration

```
property PosAxis : Integer;
```

Description

Run-time and read only.

The PosAxis property returns the corresponding Axis position in logical coordinate. This position corresponds to an horizontal coordinate for LeftAxis and RightAxis, and a vertical coordinate for TopAxis and BottomAxis components.

It is calculated when Axis line is being drawn.

It can used to detect Axis mouse clicks or to custom draw at Axis relative positions.

The TChart.ChartRect and TChart.ChartBounds properties determine the TChart rectangle and TChartAxis dimensions.

PosAxis property Example

This code draws a second Axis line of "clRed" color:

```
procedure TForm1.Chart1AfterDraw(Sender: TObject);
begin
  With Chart1,Canvas do
  begin
     Pen.Color:=clRed;
     Pen.Width:=2;
     MoveTo(ChartRect.Left,BottomAxis.PosAxis);
     LineTo(ChartRect.Right,BottomAxis.PosAxis);
  end;
end;
```

Using the TChart.ChartRect and TChartAxis coordinates properties makes same code to both draw onto Screen and Printer. Never use absolute properties:

```
BAD:  LineTo( 50, 75 );
GOOD: LineTo( ChartRect.Left + 50, ChartRect.Bottom + 75 );
```

See Also

TChart.ChartBounds

TChart.ChartRect

TChartAxis.CustomDraw

TChartAxis.CustomDrawMinMax

TChartAxis.PosLabels

TChartAxis.PosTitle

PosLabels Property

See also

Applies to

TChartAxis component

Declaration

```
property PosLabels : Integer;
```

Description

Run-time and read only.

The PosLabels property returns the corresponding Axis Labels position in logical coordinate. This position corresponds to an horizontal coordinate for LeftAxis and RightAxis Labels, and a vertical coordinate for TopAxis and BottomAxis Labels.

It is calculated when Axis labels are being drawn. Axis Labels rotation and font size do not affect this property.

It can used to detect Axis Labels mouse clicks or to custom draw at Axis Labels relative positions.

The TChart.ChartRect and TChart.ChartBounds properties determine the TChart rectangle and TChartAxis dimensions.

Please read TChartAxis.PosAxis property description for an example of use.

See Also

TChart.ChartBounds

TChart.ChartRect

TChartAxis.CustomDraw

TChartAxis.CustomDrawMinMax

TChartAxis.PosAxis

TChartAxis.PosTitle

PosTitle Property

See also

Applies to

TChartAxis component

Declaration

```
property PosTitle : Integer;
```

Description

Run-time and read only.

The PosTitle property returns the corresponding Axis Labels position in logical coordinate. This position corresponds to an horizontal coordinate for LeftAxis and RightAxis Labels, and a vertical coordinate for TopAxis and BottomAxis Labels.

It is calculated when Axis Title text is being drawn. Axis Title rotation and font size do not affect this property.

It can used to detect Axis Labels mouse clicks or to custom draw at Axis Labels relative positions.

The TChart.ChartRect and TChart.ChartBounds properties determine the TChart rectangle and TChartAxis dimensions.

Please read TChartAxis.PosAxis property description for an example of use.

See Also

TChart.ChartBounds

TChart.ChartRect

TChartAxis.CustomDraw

TChartAxis.CustomDrawMinMax

TChartAxis.PosAxis

TChartAxis.PosLabels

PrepareCanvas Method

See also

Applies to
TSeriesPointer component

Declaration

```
procedure PrepareCanvas(ColorValue : TColor);
```

Description

The PrepareCanvas method arranges all internal Canvas properties like Pen, Brush and background color to be ready to draw the Series pointer.

The ColorValue parameter is used as the pointer background color.

This method is internally called by many Series types.

See Also
TSeriesPointer.Draw

PreviousPage Method

See also

Applies to

TChart, TDBChart components

Declaration

```
procedure PreviousPage ;
```

Description

When MaxPointsPerPage is greater than Zero, TeeChart automatically divides point values in Pages. Calling PreviousPage is the same as Page := Page - 1 The NumPages chart property returns the total number of pages.

See Also

MaxPointsPerPage

ScaleLastPage property

NextPage

Print Method

See also

Applies to

TChart, TDBChart components

Declaration

```
procedure Print ;
```

Description

This method will send the Chart to the currently selected Printer. The current Printer orientation (Portrait or Landscape) can be changed prior to call PrintRect.

See Also

PrintPortrait

PrintOrientation

PrintLandscape

PrintRect

PrintPartial

Draw

PrintLandscape Method

See also

Applies to

TChart, TDBChart components

Declaration

```
procedure PrintLandscape;
```

Description

This method will set the current Printer orientation to Landscape (Horizontal paper). Then, the default Chart.Print method will be called and the previous Printer orientation will be restored.

See Also

PrintPortrait

PrintOrientation

Print

PrintRect

PrintPartial

Draw

PrintMargins Example

This code prints a Chart at Top / Right paper corner:

```
With Chart1 do
begin
  PrintMargins.Left := 60 ;
  PrintMargins.Top := 5 ;
  PrintMargins.Right:= 10 ;
  PrintMargins.Bottom:= 75 ;
  PrintLandscape;
end;
```

PrintMargins Property

See also Example

Applies to

TChart component

Declaration

```
property PrintMargins : TRect;
```

Description

Default (15, 15, 15, 15)

Run-time and read only. The PrintMargins property defines the Left, Top, Right and Bottom printer paper margins. These margins are expressed in percent of paper dimensions.

See Also

TChart.Print

TChart.PrintRect

TChart.PrintResolution

PrintOrientation Method

See also

Applies to

TChart, TDBChart components

Declaration

```
procedure PrintOrientation ( AOrientation : TPrinterOrientation ) ;
```

Description

This method will send the Chart to the currently selected Printer forcing the paper orientation to be AOrientation. The old Printer orientation is restored after printing.

See Also

PrintPortrait

PrintLandscape

PrintRect

PrintPartial

Draw

PrintPartial Method

See also

Applies to

TChart, TDBChart components

Declaration

```
procedure PrintPartial ( Const R : TRect ) ;
```

Description

This method will send the Chart to the currently selected Printer, scaled to the R rectangular region. The current Printer orientation (Portrait or Landscape) can be changed prior to call PrintRect. The biggest rectangular region is the (0,0,Printer.PageWidth-1,Printer.PageHeight-1)

The main difference between PrintPartial and PrintRect is that the first one does not call the Printer BeginDoc and EndDoc methods.

See Also

PrintPortrait

PrintOrientation

PrintLandscape

PrintRect

Draw

PrintPartialCanvas Method

See also Example

Applies to

TChart component

Declaration

```
procedure PrintPartialCanvas( PrintCanvas: TCanvas; Const PrinterRect: TRect);
```

Description

The PrintPartialCanvas method can be used to send a Chart to the Printer device. This method assumes the Printer job has been started and do not ejects the Printer page. You can determine the printed chart position and dimensions in the PrinterRect parameter.

This method allows printing more than one chart on the same page, or printing charts, custom text and custom drawings.

The TChart.PrintResolution property controls how much "wysiwyg" printing will be applied.

PrintPartialCanvas method Example

This code prints a chart on an already started printed job:

```
Chart1.PrintPartialCanvas(Printer.Canvas, Rect( 200, 200, 1000, 1000));
```

There are many other methods to print charts. See "See Also".

See Also

TChart.Print

TChart.PrintLandscape

TChart.PrintOrientation

TChart.PrintPartial

TChart.PrintPartialCanvasToScreen

TChart.PrintPortrait

TChart.PrintRect

TChart.PrintResolution

PrintPartialCanvasToScreen Method

See also

Applies to
TChart component

Declaration

```
procedure PrintPartialCanvasToScreen( PrintCanvas: TCanvas; Const
ScreenRect,PaperRect: TRect);
```

Description
The PrintPartialCanvasToScreen method allows drawing charts to screen canvases as they would look on a printer device.

The ScreenRect parameter defines the destination rectangle coordinates in pixels.

The PaperRect parameter defines virtual rectangle coordinates in printer units.

This method is used internally by the Print Preview Dialog to output chart contents to an imaginary "paper" drawn on screen.

There's currently a problem due to using Delphi canvases and anisotropic canvas mode that makes Windows GDI resources to decrease until the application is closed.

See Also

TChart.Print

TChart.PrintLandscape

TChart.PrintOrientation

TChart.PrintPartial

TChart.PrintPartialCanvas

TChart.PrintPortrait

TChart.PrintRect

TChart.PrintResolution

PrintPortrait Method

See also

Applies to

TChart, TDBChart components

Declaration

```
procedure PrintPortrait ;
```

Description

This method will set the current Printer orientation to Portrait (Vertical paper). Then, the default Chart.Print method will be called and the previous Printer orientation will be restored.

See Also

PrintLandscape

PrintOrientation

Print

PrintRect

PrintPartial

Draw

PrintRect Method

See also

Applies to

TChart, TDBChart components

Declaration

```
procedure PrintRect ( Const R : TRect ) ;
```

Description

This method will send the Chart to the currently selected Printer, scaled to the R rectangular region. The current Printer orientation (Portrait or Landscape) can be changed prior to call PrintRect. The biggest rectangular region is the (0,0,Printer.PageWidth-1,Printer.PageHeight-1)

See Also

PrintPortrait

PrintOrientation

PrintLandscape

PrintPartial

Draw

PrintResolution Property

See also

Applies to

TChart component

Declaration

```
property PrintResolution : Integer
```

Description

Default 0

Run-time and read only.The PrintResolution property controls how screen pixels will be mapped to printer pixels. By default it's zero, meaning screen proportions will be respected when printing. To get smaller fonts and thinner lines, you should set PrintResolution to a negative number.

```
Chart1.PrintResolution := 0 ; { this will use screen resolution }
Chart1.PrintResolution := -100 ; { this will use more printer resolution }
```

See Also

TChart.Print

TChart.PrintMargins

PrintTeePanel Global Variable

See also

Applies to
Global

Declaration
```
property PrintTeePanel : Boolean;
```

Description
Default False

This variable controls if Chart background panel color will be used when printing.

By default Chart panel is White when printing.

It affects all Chart components as it's a global variable.

See Also
TChart.Print

Printing Property

See also

Applies to

TChart component

Declaration

```
property Printing : Boolean;
```

Description

Run-time and read only. The Printing property indicates Chart is being printed.

See Also
TChart.Print

QRTee Unit

The QRTee unit contains the declarations for the following components and for the enumerated type associated with them. When you add a component declared in this unit to a form, the unit is automatically added to the uses clause of that form's unit. The following items are declared in the this unit:

Components

TQRChart

To see a listing of items declared in this unit including their declarations, use the Project Browser.

RadiusValues Property

See also

Applies to

TBubbleSeries component

Declaration

```
property RadiusValues : TChartValueList;
```

Description

The RadiusValues property is a TList object that stores each Bubble point Radius value. You can change Radius values by using the RadiusValues.Value[] array of doubles property:

```
BubbleSeries1.RadiusValues.Value[0] := 45.1 ;
```

This sets the first bubble point radius to 45.1

Use AddBubble and Delete methods to add and delete points.

See Also

TBubbleSeries.AddBubble

TChartValueList

ReCalcWidthHeight Method

See also

Applies to

TChart component

Declaration

```
procedure ReCalcWidthHeight;
```

Description

The ReCalcWidthHeight method recalculates the ChartWidth and ChartHeight variables.

You should maybe never call this method. It's called automatically by the TChart component.

This is the implementation of ReCalcWidthHeight:

```
procedure TChart.RecalcWidthHeight;
Begin
  ChartWidth:=ChartRect.Right-ChartRect.Left;
  if ChartWidth<=0 then ChartWidth:=1;
  ChartHeight:=ChartRect.Bottom-ChartRect.Top;
  if ChartHeight<=0 then ChartHeight:=1;
end;
```

See Also

TChart.ChartHeight

TChart.ChartWidth

RecalcMinMax Method

Applies to
TChartValueList component

Declaration
```
procedure RecalcMinMax;
```

Description
This procedure will recalculate the Axis Minimum and Maximum values. These are the Minimum of the dependent Series Minimum values and the Maximum of the dependent Series Maximum values.

RecalcOptions Property

See also

Applies to

TChartSeries component

Declaration

```
property RecalcOptions : TSeriesRecalcOptions
```

Description

Default [rOnDelete, rOnModify, rOnInsert, rOnClear];

Run-time and read only.The RecalcOptions set property controls the events that would force a recalculation of Series point values. This applies only to Series for which DataSource property is another Series. Each time one of this events happens, all depending Series are emptied and assigned again all of it's DataSource Series values. You can choose which events would force recalculation by using this property appropiately.

See the teedemo.dpr example project under "Moving Averages" (movinave.pas) unit.

See Also
TChart.CheckDatasource

RectLegend Property

See also

Applies to

TChartLegend component

Declaration

```
property RectLegend : TRect;
```

Description

Read-only and run time.

The RectLegend property returns the Legend coordinates and dimensions. It's read-only and can only be accessed at run-time. You can use RectLegend to, for example, custom draw on a chart based on Legend position coordinates or dimensions.

See Also

TChartLegend.Alignment

TChartLegend.HorizMargin

TChartLegend.TopPos

TChartLegend.VertMargin

RefreshData Method

See also

Applies to

TDBChart component

Declaration

```
procedure RefreshData;
```

Description

The RefreshData method forces TDBChart to retrieve again all Series points from their associated DataSets.

You associate DataSets (Tables, Queries, etc) to Series by setting the Series DataSource property.

It calls the RefreshDataSet method for all Series with DataSets as Series DataSources.

It do not closes nor opens nor refreshes the DataSets.

It only retrieves again all records and adds all Series points. The current DataSet filter is preserved. The current record position is saved and restored after loading all point by using a TBookMark internal variable.

For each record, you can optionally use the OnProcessRecord event to stop adding more points and retrieving more records.

When refreshing datasets, the mouse cursor can be automatically to a glass cursor by using the TDBChart.ShowGlassCursor property.

See Also

TDBChart.AutoRefresh

TDBChart.OnProcessRecord

TDBChart.RefreshDataSet

TDBChart.RefreshInterval

RefreshDataSet Method

See also

Applies to

TDBChart component

Declaration

```
procedure RefreshDataSet(ADataSet: TDataSet; ASeries: TChartSeries);
```

Description

The RefreshDataSet method will atempt to retrieve all records from the ADataSet parameter and add all points to ASeries parameter.

You associate DataSets (Tables, Queries, etc) to Series by setting the Series DataSource property.

This method works only with Active Series, that is, ASeries.Active must be True.

It only retrieves again all records and adds all Series points. The current DataSet filter is preserved. The current record position is saved and restored after loading all point by using a TBookMark internal variable.

For each record, you can optionally use the OnProcessRecord event to stop adding more points and retrieving more records.

When refreshing datasets, the mouse cursor can be automatically to a glass cursor by using the TDBChart.ShowGlassCursor property.

See Also

TDBChart.AutoRefresh

TDBChart.OnProcessRecord

TDBChart.RefreshData

TDBChart.RefreshInterval

RefreshInterval Property

See also

Applies to
TDBChart component

Declaration
```
property RefreshInterval : LongInt;
```

Description
Default Value: 0

The RefreshInterval property defines the number of seconds TDBChart will take to refresh all Datasets.

By default is zero, meaning no refresh will occur.

When greater than zero, TDBChart installs an internal TTimer component. Every time the timer expires TDBChart refreshes all Datasets and retrieves again all records.

The AutoRefresh property must be True for TDBChart to retrieve all record values.

This allows automatic real-time charting of database values.

It only happens at run-time.

See Also
TDBChart.AutoRefresh

RefreshSeries Method

See also

Applies to

TChartSeries component

Declaration

```
procedure RefreshSeries;
```

Description

The RefreshSeries method notifies all dependent Series to recalculate their points again. Each Series has a DataSource property. When DataSource is a valid Series or DataSet component, Series get all point values from the DataSource and adds them as Series points. The RefreshSeries method forces the Series to Clear and get all points again from the DataSource component. The Refreshing process traverses the Series tree recursively.

See Also

TChartSeries.DataSource

TChartSeries.Clear

TChartSeries.AddY

RemoveAllSeries Method

See also

Applies to

TChart component

Declaration

```
procedure RemoveAllSeries;
```

Description

The RemoveAllSeries method removes all Series in the Chart SeriesList. The removed Series are not freed.

This is the implementation of RemoveAllSeries method:

```
Procedure TChart.RemoveAllSeries;
Begin
 While SeriesList.Count>0 do RemoveSeries(Series[0]);
End;
```

See Also

TChart.RemoveSeries

TChart.SeriesList

RemoveSeries Method

See also

Applies to

TChart component

Declaration

```
procedure RemoveSeries(ASeries : TChartSeries);
```

Description

The RemoveSeries method deletes (but not destroys) the specified ASeries parameter from the Chart list of series.

Calling this method is exactly the same as doing:

```
ASeries.ParentChart := nil ;
```

See Also

TChart.ParentChart

TChart.RemoveAllSeries

TChart.SeriesList

Repaint Method

Applies to
TChartSeries component

Declaration
```
procedure Repaint;
```

Description
This Series method forces the whole Parent Chart to Repaint. You don't normally call Repaint directly. It can be used within derived TChartSeries components when changing internally their properties.

ResizeChart Property

See also

Applies to

TChartLegend component

Declaration

```
property ResizeChart : Boolean
```

Description

Default True

The ResizeChart property indicates if Legend will automatically reduce the Chart rectangle to prevent overlap between Legend and Chart rectangles. When ResizeChart is True, the Legend.HorizMargin and Legend.VertMargin properties control the amount of pixels that Chart rectangle will be reduced.

See Also

TChartLegend.Alignment

TChartLegend.ColorWidth

TChartLegend.HorizMargin

TChartLegend.TopPos

TChartLegend.VertMargin

RightAxis Property

See also

Applies to

TChart component

Declaration

```
property RightAxis : TChartAxis;
```

Description

The RightAxis property determines the Labels and formatting attributes of Right Chart side. It also controls where Series points will be placed.

Every TChart component has five TChartAxis: Left, Top, Right, Bottom and Depth (Z).

The RightAxis is pre-defined to be:

```
Horizontal := False ;
OtherSide := True ;
```

Refer to TChartAxis help topic for a complete description.

See Also

TChart

TChart.BottomAxis

TChart.DepthAxis

TChart.LeftAxis

TChart.TopAxis

TChartAxis

Rotate Method

See also

Applies to

TCircledSeries component

Declaration

```
procedure Rotate( Angle : Integer );
```

Description

The Rotate method adds the given Angle units to the CircledSeries.RotationAngle property. You can specify positive or negative Angle values. Positive values rotate anti-clockwise. Negative values result in clock-wise rotation. Angles must be expressed in degrees from 0 to 360.

See Also

TCircledSeries.RotationAngle

TCircledSeries.XRadius

TCircledSeries.YRadius

TCircledSeries.AngleToPos

TCircledSeries.PointToAngle

RotationAngle Property

See also

Applies to

TCircledSeries, TPieSeries components

Declaration

```
property RotationAngle : Integer;
```

Description

The RotationAngle can be a valid integer number between 0 and 359. This will the angle each Pie sector will be rotated counter-clockwise.

See Also

TCircledSeries.Rotate

TCircledSeries.XRadius

TCircledSeries.YRadius

TCircledSeries.AngleToPos

TCircledSeries.PointToAngle

RoundFirstLabel Example

Given an Axis with Minimum := 70 and Maximum := 585 :

```
Chart1.LeftAxis.RoundFirstLabel := True ;
```

Axis labels: 100 200 300 400 500

```
Chart1.LeftAxis.RoundFirstLabel := False ;
```

Axis labels: 85 185 285 385 485 585

RoundFirstLabel Property

See also Example

Applies to

TChartAxis component

Declaration

```
property RoundFirstLabel : Boolean
```

Description

Default True

Run-time only. The RoundFirstLabel property controls if Axis labels will be automatically "rounded" to the nearest magnitude. This applies both to DateTime and non-DateTime axis values. When False, Axis labels will start at Maximum Axis value.

See Also

TChartAxis.Increment

TChartAxis.Maximum

TChartAxis.Minimum

SameClass Method

Applies to
TChartSeries component

Declaration
```
Function SameClass (tmpSeries:TChartSeries):Boolean;
```

Description
(Advanced)

The SameClass method returns if any two given Series components are of the same class or derive from a common ancestor different than TChartSeries. It is used by TCustomBarSeries in stacked mode when determining which Series should be considered stackable. This allows derived Bar series types to be stacked together with normal TBarSeries instances.

SaveToBitmapFile Method

See also

Applies to

TChart, TDBChart components

Declaration

```
procedure SaveToBitmapFile( Const FileName : String ) ;
```

Description

This method will save the current chart image to the specified File Name. You should pass a valid path and name, ending with the BMP extension.

See Also

TChart.SaveToMetafileEnh

TChart.SaveToMetafile

SaveChartToFile Method

See also

Uses
teestore

Applies to
TChart, TDBChart components

Declaration
```
procedure SaveChartToFile(AChart:TCustomChart; Const AName:String);
```

Description
This method will save the current chart as a TeeChart 'tee' template to the specified File Name. Tee templates are an efficient way to save runtime Chart appearance and may be loaded at runtime using the LoadChartFromFile.

See Also

TChart.SaveToMetafileEnh

TChart.SaveToBitmapfile

LoadChartFromFile

LoadChartFromURL

TChart.SaveToMetafile

LoadChartFromURL Method

See also

Uses

teestore

Declaration

```
procedure LoadChartFromURL(Var AChart:TCustomChart; Const URL:String);
```

Description

This method will import the TeeChart 'tee' template Chart from the specified URL, applying all properties of the template to the Chart. Tee templates are an efficient way to save and load runtime Chart appearance and may be saved at runtime using the TChart.SaveChartToFile.

See Also

LoadChartFromFile

TChart.SaveToMetafileEnh

TChart.SaveToBitmapfile

TChart.SaveChartToFile

TChart.SaveToMetafile

LoadChartFromFile Method

See also

Uses

teestore

Declaration

```
procedure LoadChartFromFile(Var AChart:TCustomChart; Const AName:String);
```

Description

This method will import the TeeChart 'tee' template Chart from the specified File Name, applying all properties of the template to the Chart. Tee templates are an efficient way to save and load runtime Chart appearance and may be saved at runtime using the TChart.SaveChartToFile.

See Also

LoadChartFromURL

TChart.SaveToMetafileEnh

TChart.SaveToBitmapfile

TChart.SaveChartToFile

TChart.SaveToMetafile

SaveToMetaFile Example

This code creates a new WMF metafile:

```
Chart1.SaveToMetafile( 'c:\mychart.wmf' );
```

SaveToMetaFile Method

See also Example

Applies to

TChart component

Declaration

```
procedure SaveToMetafile(Const FileName : String);
```

Description

The SaveToMetafile method creates a new WMF file and stores Chart drawing instructions on it. WMF's are created to be "placeable" and prefixed with the Aldus header. That means the image can be resized when embedded in container applications like MS Word. The Metafile format has the advantadge of being smaller than equivalent saved images in bitmap format. Disadvantages are that some Windows GDI (graphic API) calls can't be used with metafiles. Clipping points from chart bounds is not possible with metafiles.

The new EMF format (using the 32bit version) promises to be a much better format supporting more graphic instructions.

See Also

TChart.TeeCreateMetafile

TChart.SaveToMetafileEnh

TChart.SaveToBitmapFile

SaveToMetaFileEnh Example

This code creates a new Enhanced WMF metafile:

```
Chart1.SaveToMetafileEnh( 'c:\mychart.wmf' );
```

SaveToMetaFileEnh Method

See also Example

Applies to

TChart component

Declaration

```
procedure SaveToMetafileEnh(Const FileName : String);
```

Description

The SaveToMetafileEnh method creates a new Enhanced WMF file and stores Chart drawing instructions on it. WMF's are created to be "placeable" and prefixed with the Aldus header. That means the image can be resized when embedded in container applications like MS Word. The Metafile format has the advantadge of being smaller than equivalent saved images in bitmap format. Disadvantages are that some Windows GDI (graphic API) calls can't be used with metafiles. EMF format (using the 32bit version) is a much better format supporting more graphic instructions than ‘simple’ metafile format.

See Also

TChart.TeeCreateMetafile

TChart.SaveToMetafile

TChart.SaveToBitmapFile

ScaleLastPage Property

See also

Applies to

TChart component

Declaration

```
property ScaleLastPage : Boolean
```

Description

Default True

The ScaleLastPage property controls how will be the last Chart page displayed. It only has effect when TChart.MaxPointsPerPage property is greater than zero. When True, the last Chart page will have the same horizontal scaling than the other pages. When False, the last Chart page scaling will be adjusted based on the number of visible points on that last page.

See Also

TChart.MaxPointsPerPage

NextPage

PreviousPage

Scroll method example

This button right scrolls 3 points on a ‘zoomed’ DBChart

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
 DBChart1.BottomAxis.Scroll(3,False);
end;
```

Scroll Method

See also Example

Applies to

TChartAxis component

Declaration

```
procedure Scroll(Const Offset:Double; InsideLimits:Boolean);
```

Description

This method will "scroll" or displace the Axis Maximum and Minimum values by the Offset parameter. If you want to scroll the Axis outside Series limits, InsideLimits must be False.

See Also
Scrolling at runtime

Scroll Method (TChartValueList)

Applies to
TChartValueList component

Declaration
```
procedure Scroll;
```

Description
This procedure will Left-Scroll the List of values. All values are scrolled one position to the lower point index. The first value is moved to be the Last value in the List.

Series Axis Dependence

By default, each Series is assigned to depend on Left and Bottom Axis. You can set which Horizontal and Vertical axis will depend on each Series. By default, all axis are set to be Automatic. That means TeeChart will calculate the Maximum and Minimum values for each axis based on which Series are set to depend on it.

For example, if you want to mix both DateTime and NON-DateTime series in the same chart, you will need to assign a different axis to the series :

Example:

```
LineSeries1.XValues.DateTime := True ;
LineSeries1.HorizAxis    := aBottomAxis ;
LineSeries2.XValues.DateTime := False ;
LineSeries2.HorizAxis    := aTopAxis ;
```

This will scale the X Series values accordingly.

You can also do the same for the vertical axis using the Series VertAxis property:

```
LineSeries1.VertAxis := aLeftAxis ;
LineSeries2.VertAxis := aRightAxis ;
```

You can control if Chart Axis will be scaled automatically or you can set the Axis Maximum, Minimum and Increment manually.

If you allow users to Zoom or Scroll the Chart, then the Axis will be scaled or scrolled.

Series Example

The following example will set all Bar Series Style to Cilinders

```
for t := 0 to Chart1.SeriesCount - 1 do
  if Chart1.Series[ t ] is TBarSeries then
     ( Chart1.Series[ t ] as TBarSeries ).BarStyle := bsCilinder ;
```

The following example will count all Active Series points

```
counter := 0 ;
for t := 0 to Chart1.SeriesCount - 1 do
  counter := counter + Chart1.Series[ t ].Count ;
```

Series Property

Example

Applies to

TChart, TDBChart components

Declaration

```
property Series[Index:Longint]:TChartSeries
```

Description

The Series array property is the list of Series that the Chart component maintains and draws. (Series can be Active or not).

Series Unit

The Series unit contains the declarations for the following components and for the enumerated type associated with them. When you add a component declared in this unit to a form, the unit is automatically added to the uses clause of that form's unit. The following items are declared in the this unit:

Series are truly invisible components as they don’t show on the Form. At design time you can access Series components using the Object Inspector or double-clicking on the chart to access the Chart Editor.

See the TeeChart Gallery, accessible via the Chart Editor for a full visual display of Series types.

Click here to see the included TeeChart Series types.

For a list of the variables that each Series type supports see the variables list.

Components

TAreaSeries

TBarSeries

TCircledSeries

TCustomBarSeries

TCustomSeries

TFastLineSeries

THorizBarSeries

TLineSeries

TPieSeries

TPointSeries

TSeriesPointer

Types

TBarStyle

TGetBarStyleEvent

TMultiArea

TMultiBar

TSeriesClickPointerEvent

TSeriesPointerStyle

Routines

ApplyDark Method

To see a listing of items declared in this unit including their declarations, use the Project Browser.

SeriesColor Example

```
LineSeries3.SeriesColor := clBlue ;
AreaSeries1.SeriesColor := clYellow ;
```

SeriesColor Property

See also Example

Applies to

TChartSeries component

Declaration

```
property SeriesColor : TColor
```

Description

Default color is clTeeColor.

The TChartSeries SeriesColor property is the default color in which the Series points will be drawn on. This property value could be any valid Delphi or Windows color. If you add points with clTeeColor color, then they will be drawn with the SeriesColor color. This property is the default Color associated to the Series.

When you place a new Series component in a Chart, TeeChart will assign a free color to this property (a Color that no other Series in the same Chart use). Some Series have the ColorEachPoint boolean property. Setting this to TRUE will force the Series to paint each point with a different color, thus without using its SeriesColor

SeriesColor is also used to paint the small rectangle in TChartLegend.

See Also
ColorEachPoint

SeriesCount Example

```
for t := 0 to Chart1.SeriesCount - 1 do
  Chart1.Series[ t ].SeriesColor := clRed ;
```

SeriesCount Method

Example

Applies to

TChart, TDBChart components

Declaration

```
function SeriesCount : Longint ;
```

Description

This is the number of Series (Active or not) a Chart has.

SeriesDown Method

Applies to
TChart, TDBChart components

Declaration
```
procedure SeriesDown ( ASeries : TChartSeries ) ;
```

Description
The SeriesUp method will "send to back" the specified ASeries. That will change the order the Series are drawn.

WARNING:

Series order is not saved to the DFM form file.

SeriesHeight3D Property

See also

Applies to

TChart component

Declaration

```
property SeriesHeight3D : Longint;
```

Description

Run-time and read only. The SeriesHeight3D property determines the default Series height when drawing in 3D mode. The Chart.View3D property should be True. SeriesHeight3D changes when resizing a Chart, when changing the number of visible Series or when changing the Chart3DPercent property.

See Also

TChart.View3D

TChart.Chart3DPercent

TChart.SeriesWidth3D

TChart.Width3D

TChart.Height3D

TChartSeries.ZOrder

SeriesList Property

See also

Applies to

TChart, TDBChart components

Declaration

```
property SeriesList : TChartSeriesList;
```

Description

Run-time only. The SeriesList property is an array property that contains a list of pointers to all Series linked to a TChart component.

You DO NOT NEED to use this property to traverse all Series and do specific things with them:

```
for t:= 0 to Chart1.SeriesCount -1 do
with Chart1.Series[ t ] do
begin
 SeriesColor := clRed ;
end;
```

So, YOU NEED this property to, for example, change the Series ordering at run-time. This allows control of Series Z Order positioning:

```
Chart1.SeriesList[ 0 ] := MySeries2 ;
Chart1.SeriesList[ 1 ] := MySeries1 ;
Chart1.SeriesList[ 2 ] := MySeries3 ;
```

Warning: Never FREE or REMOVE SeriesList elements. Use the Series.Free to remove it or use the Series.Active property to disable it.

See Also

TChart.Series

TChart.SerisCount

TChartSeries.Delete

TChartSeries.Active

SeriesTitleLegend Example

That will assign the first Chart Series Title to a label caption

```
Label1.Caption := Chart1.SeriesTitleLegend( 0, False ) ;
```

That will assign the first ACTIVE Chart Series Title to a label caption

```
Label1.Caption := Chart1.SeriesTitleLegend( 0, True ) ;
```

SeriesTitleLegend Method

Example

Applies to

TChart, TDBChart components

Declaration

```
function SeriesTitleLegend ( SeriesIndex : Longint ; ActiveOnly :
Boolean ) : String ;
```

Description

The SeriesTitleLegend function returns the Series.Title string.

SeriesUp Method

See also

Applies to

TChart, TDBChart components

Declaration

```
procedure SeriesUp ( ASeries : TChartSeries ) ;
```

Description

The SeriesUp method will "bring to front" the specified ASeries. This will change the order in which Series are drawn.

WARNING:

Series order is not saved to the DFM form file.

See Also
TChartSeries.ZOrder

SeriesWidth3D Property

See also

Applies to

TChart component

Declaration

```
property SeriesWidth3D : Longint;
```

Description

Run-time and read only. The SeriesWidth3D property determines the default Series width when drawing in 3D mode. The Chart.View3D property should be True. SeriesWidth3D changes when resizing a Chart, when changing the number of visible Series or when changing the Chart3DPercent property.

See Also

TChart.Chart3DPercent

TChart.Height3D

TChart.SeriesHeight3D

TChart.View3D

TChart.Width3D

TChartSeries.ZOrder

SetBrushCanvas Example

We want to custom draw a rectangle filled with White color and a Vertical pattern Brush of Red color. The rectangle will cover all Chart screen space between Axis.

(Don't think this is useful, just an example... <g>)

```
Chart1.SetBrushCanvas(tmpCanvas, clRed, bsVertical, clWhite );
With Chart1.ChartRect do
  Chart1.Canvas.Rectangle( Left, Top, Right, Bottom );
```

SetBrushCanvas Method

See also Example

Applies to

TChart component

Declaration

```
procedure SetBrushCanvas(tmpCanvas: TCanvas3D;AColor: TColor; APattern:
TBrushStyle; ABackColor: TColor);
```

Description

The SetBrushCanvas method solves an annoying problem with the Delphi standard behaviour. Delphi VCL automatically sets the Windows GDI background mode to TRANSPARENT when the Brush.Style property is set to a different Brush style than bsSolid. This means all filled Screen parts should be erased previous to apply the transparent brush. This method calls the Windows GDI API directly to set the Windows Brush mode to OPAQUE and Windows Brush Background Color to ABackColor parameter. Use this method when you want to draw non solid patterns that need to be OPAQUE (erasing existing pixels before filling).

See Also

TChart.Canvas

TChart.FontCanvas

SetInternalCanvas Method

See also

Applies to

TChart, TDBChart components

Declaration

```
Procedure SetInternalCanvas(NewCanvas:TCanvas);
```

Description

(Advanced)

The SetInternalCanvas method replaces TChart internal canvas component used to draw chart contents. It can be seen as a protected Set method for TChart.Canvas property, which is read-only.

See Also
TChart.Canvas

SetMinMax Example

This call:

```
Chart1.LeftAxis.SetMinMax( 0, 100 ) ;
```

is equivalent to:

```
Chart1.LeftAxis.Automatic:= False ;
Chart1.LeftAxis.Minimum := 0 ;
Chart1.LeftAxis.Maximum := 100 ;
```

SetMinMax Method

See also Example

Applies to

TChartAxis component

Declaration

```
procedure SetMinMax( Const AMin, AMax : Double );
```

Description

The SetMinMax method changes the current Axis Minimum and Maximum scales. The TChartAxis.Automatic property is set to False.

See Also

TChartAxis.Automatic

TChartAxis.Maximum

TChartAxis.Minimum

ShadowColor Property

See also

Applies to

TChartLegend component

Declaration

```
property ShadowColor : TColor
```

Description

Default clBlack

The ShadowColor property specifies the color to fill the Legend shadow effect. You can control the Shadow effect pixels size by using the Legend.ShadowSize property.

ShadowColor Property (TPieSeries)

See also

Applies to

TPieSeries component

Declaration

```
property ShadowColor : TColor;
```

Description

The ShadowColor property determines the color used to fill the PieSeries 3D zone when Shadowed3D property is True.

The ShadowColor is used to draw many vertical lines to make Pie 3D zone darker.

See also
TPieSeries.Shadowed3D

See Also
TChartLegend.ShadowSize

ShadowSize Property

See also

Applies to

TChartLegend component

Declaration

```
property ShadowSize : Integer
```

Description

Default 3

The ShadowSize property controls the amount of pixels used to draw the Legend shadow effect. Setting this property to zero hides the Legend shadow effect. You can change the shadow color by using the Legend.ShadowColor property.

See Also
TChartLegend.ShadowColor

Shadowed3D Property

See also

Applies to

TPieSeries component

Declaration

```
property Shadowed3D : Boolean;
```

Description

Default Value: False

The Shadowed3D property determines if the 3D Pie zone will be filled with dark lines to make a dark visual effect.

See Also

TPieSeries.Color3D

TPieSeries.ShadowColor

ShowGlassCursor Property

See also

Applies to
TDBChart component

Declaration
```
property ShowGlassCursor : Boolean;
```

Description
Default Value: True

The ShowGlassCursor property determines if TDBChart will change the current cursor shape to a glass cursor whenever records are retrieved from Tables or Queries.

This notifies the user of record retrieval activity.

For fast and small queries or tables you might want to set it to False, thus not disturbing the user.

See Also

TDBChart.AutoRefresh

TDBChart.RefreshDataSet

TDBChart.RefreshInterval

ShowInLegend Property

Applies to
TChartSeries component

Declaration
```
property ShowInLegend : Boolean
```

Description
Default is true

The property controls whether or not the series title should display in Chart.Legend. It is only meaningful when when LegendStyle is lsSeries or lsLastValues.

SideMargins Property

See also

Applies to

TBarSeries and THorizBarSeries components (TCustomBarSeries)

Declaration

```
property SideMargins : Boolean
```

Description

Default True

The SideMargins property controls if first and last displayed Bar will be separated from the Chart rectangle. By default, margins are set to half the sum of all Bar Series bar widths.

See Also

TBarSeries.BarWidth

TBarSeries.CustomBarWidth

THorizBarSeries.BarHeight

THorizBarSeries.CustomBarHeight

Size3d Method

Applies to
TChart, TDBChart components

Declaration
```
procedure Size3d ( Var Horiz , Vert : Longint ) ;
```

Description
Size3d method returns in Horiz and Vert parameters the default 3d size. The 3d size depends on Chart.Chart3dPercent property.

Sort Method

See also

Applies to

TChartValueList component

Declaration

```
procedure Sort;
```

Description

The Sort method re-orders Series points interchanging their position in the Series values lists so they will be displayed in different coordinates. By default, Series are set to sort points in ascending order using their X coordinates. This is accomplished with this code:

Series1.XValues.Order := loAscending ;

Series1.YValues.Order := loNone ;

By default, Series draw points using the point ValueIndex, except in some non common situations like having the horizontal axis inverted.

Important Note: Re-Ordering Series points do NOT change point X coordinates. Series points which have no X coordinates are assigned a unique incremental number to determine the point horizontal positions. Automatic Point indexes start at zero. You will need to change every point X coordinate when sorting Series points with automatic X values.

The following code re-orders a Series by it's Y values in descending order:

Note: XValues.FillSequence should not be called when sorting XY points.

```
procedure TForm1.BitBtn3Click(Sender: TObject);
begin
 With Series1 do { <-- use a TBarSeries for example }
 begin
  FillSampleValues(8);
  XValues.Order:=loNone ;
  YValues.Order:=loDescending ;
  YValues.Sort ;
  XValues.FillSequence ;
  Repaint ;
 end;
end;
```

See Also

TChartValueList.Order

TChartValueList.FillSequence

Squared Property

See also

Applies to

TBubbleSeries component

Declaration

```
property Squared : Boolean
```

Description

Default value: True

The Squared property determines how Bubble size is calculated. By default, the horizontal and vertical Bubble sizes are equal to each Bubble's radius. When Squared is False, Bubble horizontal and vertical sizes are calculated independently based on Series axis scales.

See Also

TBubbleSeries.RadiusValues

TSeriesPointer.HorizSize

TSeriesPointer.VertSize

Stairs Property

Applies to
TLineSeries, TAreaSeries components

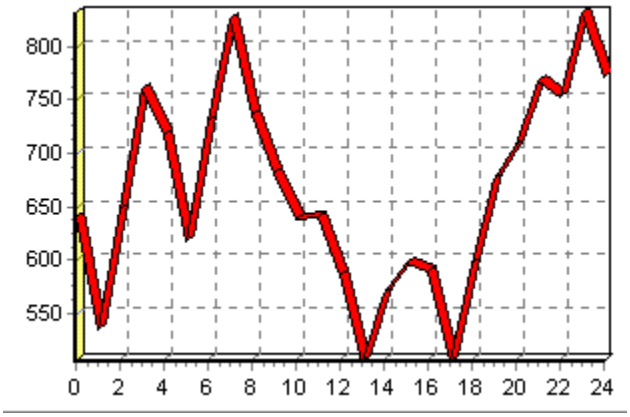
Declaration
```
property Stairs : Boolean;
```

Description
This boolean property controls the drawing of LineSeries and Areaseries.

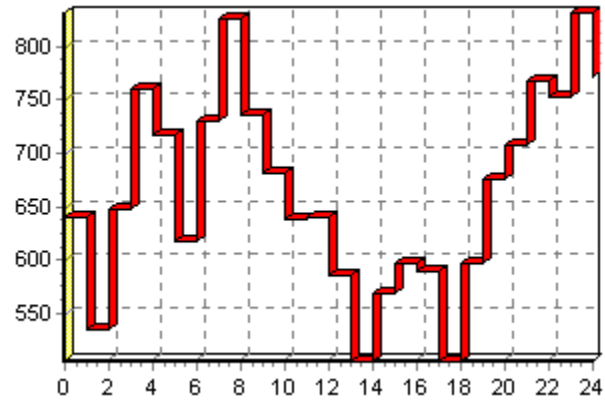
In most normal situations, a series draws a line between each Line point. This makes the Line appear as a "mountain" shape. However, setting Stairs to TRUE will make the Series to draw 2 Lines between each pair of points, thus giving a "stairs" appearance. This is most used in some financial Chart representations.

When Stairs is set to True you may set InvertedStairs to True to alter the direction of the step. - see charts below.

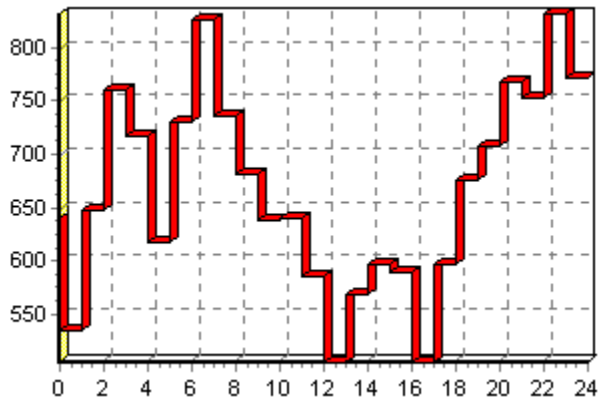
Line with Stairs False



Same line with Stairs True



Same line with Stairs True and InvertedStairs False


800
750
700
650
600
550
0 2 4 6 8 10 12 14 16 18 20 22 24

StartColor Property (TChartGradient)

See also

Applies to
TChartGradient component

Declaration
```
property StartColor : TColor;
```

Description
Default: clWhite

The StartColor property is one of the two colors used to create the gradient fill. The gradient fill is composed of two colors: StartColor and EndColor.

See also

TChartGradient.Direction

TChartGradient.EndColor

TChartGradient.Visible

StartValues Property (TGanttSeries)

See also

Applies to

TGanttSeries component

Declaration

```
property StartValues : TChartValueList;
```

Description

The StartValues property defines the starting Gantt bar date values. The ending Gantt bar point date is stored at TGanttSeries.EndValues list property.

StartValues and EndValues can be specified both as DateTime or double values.

Both are standard TChartValueList components. That means you can access their values with same methods as you can access X or Y values.

The TGanttSeries.AddGantt and / or TGanttSeries.AddGanttColor methods must be used to add Gantt bar points.

See also

TGanttSeries.AddGantt

TGanttSeries.AddGanttColor

TGanttSeries.EndValues

StartXValues Example

This code modifies the X0 coordinate of first arrow point:

```
tmp:=ArrowSeries1.StartXValues.Value[0];
ArrowSeries1.StartXValues.Value[0]:=tmp + 123;
```

StartXValues Property

See also Example

Applies to

TArrowSeries component

Declaration

```
property StartXValues : TChartValueList
```

Description

Each Arrow has (X0,Y0) and (X1,Y1) coordinates.

StartXValues property is the list of X0 values.

StartXValues.DateTime property default value is True.

See Also

TChartValueList

TArrowSeries.AddArrow

TArrowSeries.EndXValues

TArrowSeries.EndYValues

TArrowSeries.StartYValues

StartYValues Example

This code modifies the Y0 coordinate of first arrow point:

```
tmp:=ArrowSeries1.StartYValues.Value[0];
ArrowSeries1.StartYValues.Value[0]:=tmp + 123;
```

StartYValues Property

See also Example

Applies to

TArrowSeries component

Declaration

```
property StartYValues : TChartValueList
```

Description

Each Arrow has (X0,Y0) and (X1,Y1) coordinates.

StartXValues property is the list of Y0 values.

See Also

TChartValueList

TArrowSeries.AddArrow

TArrowSeries.EndXValues

TArrowSeries.EndYValues

TArrowSeries.StartXValues

StatChar Unit

The StatChar unit contains the declarations for the following components and for the enumerated type associated with them. When you add a component declared in this unit to a form, the unit is automatically added to the uses clause of that form's unit. The following items are declared in the this unit:

Components

TAverageTeeFunction

TTeeFunction

Routines

To see a listing of items declared in this unit including their declarations, use the Project Browser.

Style Property (TChartPen)

Applies to
TChartPen component

Declaration
```
property Style : TPenStyle;
```

Description
The Style property determines the style in which the pen draw lines on the canvas.

See Delphi help: TPen.Style

Style Property (TSeriesMarks)

See also

Applies to

TSeriesMarks component

Declaration

```
property Style : TSeriesMarksStyle;
```

Description

Default Value: smsLabel

The Style property defines how Series Marks are constructed. Each different Style value makes Marks to output a different text. Several options are available, but you can also use the TChartSeries.OnGetMarkText event and override the default Series Marks text.

The different Mark styles are:

smsValue

Shows the point value. It is usually the YValue, except THorizBarSeries that defaults to the XValue. The TChartSeries.ValueFormat property is used to format the resulting string.

Example: "1234"

smsPercent

Shows the percent the point value represents. The TChartSeries.PercentFormat property is used to format the percent value.

Example: "12 %"

smsLabel

Shows the associated Point Label.

Example: "Cars"

Warning:

If the point has no Label (empty label), then the point value is used. You can use TChartSeries.OnGetMarkText event to force an empty label for a particular point.

smsLabelPercent

Shows the point Label and the percent value the point represents. The TChartSeries.PercentFormat property is used to format the percent value.

Example: "Cars 12 %"

smsLabelValue

Shows the point Label and the point value. The TChartSeries.ValueFormat property is used to format the resulting string.

Example: "Cars 1234"

smsLegend

Shows whatever is shown at Chart Legend. Please refer to TChartLegend.TextStyle property for a list of possible values.

smsPercentTotal

Shows the percent the point represents toghether with the "of" word and the sum of all points absolute values. The TChartSeries.PercentFormat property is used to format the percent value.

Example: "12 % of 1234"

NOTE:

You can alter or localize to your language the "of" word by using the "PercentOf" typed constant in Teengine unit:

```
Example in Spanish: PercentOf := 'sobre';
Output:  "12 % sobre 1234"
```

smsLabelPercentTotal

Shows the point Label toghether with the resulting "smsPercentTotal" style. The TChartSeries.PercentFormat property is used to format the percent value.

Example: "Cars 12 % of 1234"

smsXValue

Shows the point XValue (same as smsValue but with X values instead of Y values).

Example: "21/6/1996"

Style Property (TSeriesPointer)

See also

Applies to

TSeriesPointer component

Declaration

```
property Style : TSeriesPointerStyle;
```

Description

Default Value: psRectangle

The Style property defines the shape used to display the Series Points.

The default psRectangle style can be optionally in 3D mode by setting the Pointer.Draw3D property to True.

The Series Pointer.Visible property should be True.

See Also

TChartSeries.MarkPercent

TChartSeries.Marks

TChartSeries.OnGetMarkText

TChartSeries.PercentFormat

TChartSeries.ValueFormat

See Also

TSeriesPointer.Draw3D

TSeriesPointer.HorizSize

TSeriesPointer.VertSize

SwapValueIndex Method

See also

Applies to

TChartSeries component

Declaration

```
procedure SwapValueIndex( a, b : Longint );
```

Description

The SwapValueIndex method interchanges two Series points. The A and B parameter should be valid point indexes, from 0 to Series Count-1.

The Chart component is automatically redrawn to display the new Series points order.

See Also
TChartSeries.Count

TAddTeeFunction component Example

To set the period for TAddTeeFunction you should use the FunctionType property of TChartSeries

To define a function series by code you should first create a new series for the function. The series may be of any type.

```
{ Set the function using the SetFunction method}.
Series1.SetFunction(TAddTeeFunction.Create(Self));
{You may then define the period for the function - here setting it to to 5}
Series1.FunctionType.Period:=5;
```

To undefine (delete) a function defined for the series use

```
Series1.SetFunction(nil);
```

TAddTeeFunction Component

See also Example

Unit

TeeFunci

Ancestor

TTeeFunction

Description

TAddTeeFunction may be added to your project by Chart Editor at design time or at runtime using code. Default period for the TAddTeeFunction when only 1 series is added to the function is 0 (One flat line representing the total of all data points of the input series). When 2 or more input series form the TAddTeeFunction series the default period is 1 axis point. Period is applicable only to the number of axis points, Period 1 = 1 axis point; period 2 = 2 axis points, etc..

To see a visual representation of TeeChart Standard Functions, go to the TeeChart User Guide.

See Also

TSubtractTeeFunction

TMultiplyTeeFunction

TDivideTeeFunction

TAverageTeeFunction

THighTeeFunction

TLowTeeFunction

FunctionType_property

TAreaSeries Component

Properties Methods Events

Unit

Series

Ancestor

TCustomSeries

Description

The TAreaSeries component outputs all points by drawing a line between them and fills the area defined by the line and the bottom side of the Chart. To see a visual representation of this Series type, go to the TeeChart User Guide.

Set the ParentChart property to the desired Chart component. Use the AddY or AddXY method to manually fill area points.

Use the AreaBrush property to draw patterned areas.

The AreaColor property determines the background area color.

The lines dividing area points are drawn using the AreaLinesPen property.

To build stacked area charts use the MultiArea property.

The Stairs property controls how to draw the connecting lines between points. Use then the InvertedStairs property to change how are these steps drawn.

The top lines can be altered by using the LinePen property.

Please refer to TCustomSeries ancestor description for all common Series properties like Marks, Axis, Pointers and events.

Methods

Key Methods

Clicked

GetOriginPos

Properties

Run-time only

Key Properties

AreaBrush

AreaColor

MultiArea

AreaLinesPen

DrawArea

LinePen

InvertedStairs

Stairs

XValues

YValues

TArrowSeries Component

Properties Methods Events

Unit

ArrowCha

Ancestor

TPointSeries

Description

The TArrowSeries component outputs all points as arrows. To see a visual representation of this Series type, go to the TeeChart User Guide.

Set the ParentChart property to the desired Chart component. Use the AddArrow method to manually fill area points.

Right-click the component at design-time to access the DataSource Wizard Dialog to connect the Series to another Series or to any Table or Query component.

The arrow head dimensions can be changed using the ArrowHeight and ArrowWidth properties.

Each arrow is defined by four coordinates. These values can be accessed using the StartXValues, StartYValues, EndXValues and EndYValues properties.

Arrows are always drawn in 2D mode.

Please refer to TPointSeries ancestor description for all common Series properties like Marks, Axis, Pointers and events.

Methods

Key Methods

AddArrow

Properties

Run-time only

Key Properties

ArrowHeight

ArrowWidth

EndXValues

EndYValues

StartXValues

StartYValues

TAverageTeeFunction component Example

To set the period for TAverageTeeFunction you should use the FunctionType property of TChartSeries

To define a function series by code you should first create a new series for the function. The series may be of any type.

```
{ Set the function using the SetFunction method}.
Series1.SetFunction(TAddTeeFunction.Create(Self));
{You may then define the period for the function - here setting it to to 5}
Series1.FunctionType.Period:=5;
```

To undefine (delete) a function defined for the series use

```
Series1.SetFunction(nil);
```

.

TAverageTeeFunction Component

See also Properties Example

Unit
TeeFunci

Ancestor
TTeeFunction

Description
TAverageTeeFunction performs a simple or weighted average of last Period DataSource values. TAverageTeeFunction Period is the number of points we want to group to calculate their average. So, if our Series has 1000 points and we have a Period of 200, then AverageTeeFunction will have 1000/200 (that is 5) points. Each Y value will be the simple Average of a group of 200 points.

To see a visual representation of TeeChart Standard Functions, go to the TeeChart User Guide.

See Also

TAddTeeFunction

TMultiplyTeeFunction

TDivideTeeFunction

TSubtractTeeFunction

THighTeeFunction

TLowTeeFunction

FunctionType_property

Properties

Run-time only

Key Properties

Weighted

TAxisLabelStyle Type

Unit
TeEngine

Declaration
```
TAxisLabelStyle = (talAuto,talNone,talValue,talMark,talText);
```

Description
TAxisLabelStyle defines the possible values of the TChartAxis.LabelStyle property.
You can override each axis label text by using the Chart.OnGetAxisLabel event.
Possible values for TChartAxis.LabelStyle are:

talAuto	Choose the Style automatically.
talNone	No label. This will trigger the event with empty strings.
talValue	Axis labeling is based on axis Minimum and Maximum properties.
talMark	Each Series point will have a Label using SeriesMarks style.
talText	Each Series point will have a Label using Series.XLabels strings.

TAxisOnGetLabel Type

Unit
TeEngine

Declaration
```
TAxisOnGetLabel = procedure(Sender:TChartAxis; Series:TChartSeries;
ValueIndex: LongInt; Var LabelText: String) of object;
```

Description
The TAxisOnGetLabel type points to a method that notifies a TChartAxis component that an event has occurred.

TAxisOnGetNextLabel Type

Unit
TeEngine

Declaration
```
TAxisOnGetNextLabel = Procedure(Sender: TChartAxis; LabelIndex: LongInt; Var
LabelValue: Double; Var Stop: Boolean) of object;
```

Description
The TAxisOnGetNextLabel type points to a method that notifies a dataset component that an event has occurred. It is used by the OnGetNextAxisLabel event.

TBarSeries Component

Properties Methods

Unit

Series

Ancestor

TCustomBarSeries

Description

The TBarSeries component outputs all points as vertical bars. To see a visual representation of this Series type, go to the TeeChart User Guide.

Several TBarSeries can be displayed side-to-side, one behind the other, stacked or stacked 100% by using the MultiBar property.

Set the ParentChart property to the desired Chart component. Use the AddBar or AddXY methods to manually fill area points.

Right-click the component at design-time to access the DataSource Wizard Dialog to connect the Series to another Series or to any Table or Query component.

The BarBrush property determines the pattern used to fill bars, while the BarPen property is used to draw the bar edges.

Set the desired bar style (cilynder, pyramid, etc) by changing the BarStyle property.

Use the BarWidthPercent property to control the relative distance between bars.

You can specify an exact bar width by using the CustomBarWidth property.

The Dark3D property controls if bar sides are filled with a darker color than front bar faces.

The OffsetPercent property determines the bars horizontal displacement. This can be used to create overlayed bars with several bar series components.

Bar series leave margins both at left and right chart sides. You can turn off this default behaviour setting the SideMargins property to False.

By default, bar bottoms start at zero vertical coordinate. Set the YOrigin property to the desired starting bottom value or set the UseYOrigin to False to make bar bottoms start at minimum bars value.

The OnGetBarStyle event can be used to supply a different bar style for each Series point.

Please refer to TCustomBarSeries ancestor description for all common Series properties like Marks, Axis dependence, methods and events.

Methods

Key Methods

AddBar

Clicked
GetOriginPos

Properties

Run-time only

Key Properties

AutoMarkPosition

BarBrush DarkColor

UseYOrigin

BarPen DarkerColor YOrigin

BarStyle

MultiBar

BarWidth

NormalBarColor

BarWidthPercent OffsetPercent

CustomBarWidth ParentChart

Dark3D SideMargins

TBarStyle Type

Unit
Series

Declaration
```
TBarStyle = (bsRectangle,bsRectGradient,
bsPyramid,bsInvPyramid,bsCilinder,bsEllipse,bsArrow);
```

Description
TBarlStyle defines the possible values of the TCustomBarSeries.BarStyle property.

These are the possible Bar styles:

bsArrow

bsCilinder

bsEllipse

bsInvPyramid

bsPyramid

bsRectangle

TBubbleSeries Component

Properties Methods

Unit

BubbleCh

Ancestor

TPointSeries

Description

The TBubbleSeries component outputs all points as "bubbles", each one with a different bubble radius value. To see a visual representation of this Series type, go to the TeeChart User Guide.

Set the ParentChart property to the desired Chart component. Use the AddBubble method to manually fill area points.

Right-click the component at design-time to access the DataSource Wizard Dialog to connect the Series to another Series or to any Table or Query component.

All radius values are stored at RadiusValues property.

The Pointer property determines all formatting attributes.

By default, the radius values are expressed in vertical axis coordinates. Set the Squared property to False to calculate bubble heights in vertical coordinates and bubble widths in horizontal coordinates.

Please refer to TPointSeries ancestor description for all common Series properties like Marks, Axis dependence, Pointers and events.

Methods

- **Key Methods**
  - AddBubble

Properties

Run-time only

Key Properties

RadiusValues Squared

TChart Component

Properties Methods Events

Unit
Chart

Ancestor
TCustomChart

Description
TChart is the most important component in TeeChart library. TChart derives from TPanel and inherits all its functionality. In short, TChart is a standard Delphi TPanel component with many new capabilities specific to charting and graphing purposes.

TChart is the ParentChart component for Series.

- You can create Chart components both at design and run-time.
- Many charts can exist in a Form.
- TeeChart components can be also in an ActiveX Form.

For an indepth look at how to create Charts with TeeChart see the TeeChart User Guide.

TChart Events

Key Events

OnAfterDraw
OnAllowScroll
OnClick
OnClickAxis
OnClickBackground
OnClickLegend
OnClickSeries
OnDblClick
OnGetAxisLabel
OnGetLegendPos
OnGetLegendRect
OnGetLegendText
OnGetNextAxisLabel
OnMouseDown
OnMouseUp
OnPageChange
OnResize
OnScroll
OnUndoZoom
OnZoom

Language Support

TEECONST.RES

Now TeeChart includes a resource file that's linked with TeeChart units. This resource file contains the internal exception strings and warning messages.

In case you want to translate it to any other language, you can use the Borland Resource Workshop editor to access and change the string contents.

If you consider a new language would be useful for other TeeChart'ers, feel free to email it to TeeChart support.

TChart Methods

Key Methods

ActiveSeriesLegend
AddSeries
Assign
BackWallRect
CalcClickedpart
CalcSize3d
CalcSize3dWalls
CanvasChanged
ChartPrintRect
ChartRegionRect
ChartXCenter
ChartYCenter
CheckDatasource
CopyToClipBoardBitmap
CopyToClipBoardMetafile
ExchangeSeries
FontCanvas
FormattedLegend
FormattedValueLegend
GetASeries
GetAxisSeries
GetCursorPos
GetFreeSeriesColor
GetLabelsSeries
GetRectangle
GetWidthHeight
IsFreeSeriesColor
IsScreenHighColor
IsValidDataSource
MarkText
MaxMarkWidth
MaxTextWidth
MaxXValue
MaxYValue
MinXValue
MinYValue
NextPage
NumPages
Print
PrintLandscape
PrintOrientation
PrintPartialCanvas
PrintPartialCanvasToScreen
PrintPartial
PrintPortrait
PrintRect
ReCalcWidthHeight
RemoveAllSeries
RemoveSeries
RotateLabel
SaveChartToFile
SaveToBitmapFile
SaveToMetafileEnh
SaveToMetafile
SeriesCount
SeriesDown
SeriesTitleLegend
SeriesUp
SetInternalCanvas
Size3d
TeeCreateMetafile
UndoZoom
XLabelText
ZoomPercent
ZoomRect

TChart Properties

Run-time only

Key Properties

AllowPanning
AllowZoom
AnimatedZoom
AnimatedZoomSteps
AxisVisible
Color
Foot
Frame
Gradient
Printing
PrintMargins
PrintResolution
RightAxis

Height3D
BackColor
BackImage
BackImageInside
BottomAxis
BackWall
BottomWall
BufferedDisplay
CancelMouse
LeftAxis
LeftWall
Legend
MarginBottom
DepthAxis
MarginLeft
MarginRight
MarginTop
ScaleLastPage
Series
SeriesHeight3D
SeriesList
SeriesWidth3D
View3DOptions
Title
TopAxis
View3d

Canvas

MaxPointsPerPage
Chart3dPercent
View3dWalls
MaxZOrder

Width3D
ChartBounds
ChartHeight
ChartRect
ChartWidth
Monochrome
MonochromePrinting
OriginalCursor

BackImage
ClipPoints
BackImageMode

TChartAxis Component

Properties Methods

Unit

TeEngine

Description

TChartAxis is a Chart subcomponent.

Chart components have 5 TChartAxis:

LeftAxis, RightAxis, TopAxis, BottomAxis and DepthAxis.

Methods

Key Methods

AdjustMaxMin
CalcIncrement
CalcLabelStyle
CalcMinMax
CalcPosPoint
CalcPosValue
CalcRect
CalcSizeValue
CalcXPosValue
CalcXSizeValue
CalcXYIncrement
CalcYPosValue
CalcYSizeValue
Clicked
CustomDraw
CustomDrawMinMax
CustomDrawMinMaxStartEnd
CustomDrawStartEnd
DrawAxisLabel
DrawHorizontalLabel
Draw
DrawVerticalLabel
Horizontal
SizeTickAxis
SizeTitle
IsDateTime
IsDepthAxis
LabelHeight
LabelValue
LabelWidth
MaxLabelsWidth
OtherSide
Scroll
SetMinMax
SizeLabels

Properties

Run-time only

Key Properties

Automatic
LabelsFont

PosAxis

AutomaticMaximum
LabelsOnAxis

PosLabels

AutomaticMinimum
LabelsSeparation

PosTitle

Axis
LabelsSize
RoundFirstLabel

AxisValuesFormat
LabelStyle
TickInnerLength

DateTimeFormat
Logarithmic
TickLength

ExactDateTime
Maximum
TickOnLabelsOnly

Grid
Minimum
Ticks

Horizontal
MinorTickCount
TicksInner

Increment
MinorTickLength
Title

Inverted
MinorTicks
TitleSize

Labels

OtherSide
Visible

LabelsAngle
ParentChart
LabelsMultiLine

GridCentered
IStartPos
IEndPos

PositionPercent
StartPosition
EndPosition

TChartAxisTitle Component

Properties

Unit

TeEngine

Description

The Axis will use this to draw its Title.

Properties

Run-time only

Key Properties

Angle

Caption

Font

TChartClick Type

Unit
Chart

Declaration
```
TChartClick = procedure(Sender: TCustomChart; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer) of object;
```

Description
The TChartClick type points to a method that notifies a TChart component that a click event has occurred. It is used by OnClickLegend and OnClickBackground

TChartClickAxis Type

Unit
Chart

Declaration
```
TChartClickAxis = procedure( Sender : TCustomChart; Axis : TChartAxis;
Button:TMouseButton; Shift: TShiftState; X, Y: Integer) of object;
```

Description
The TChartClickAxis type points to a method that notifies a TChartAxis component that a click event has occurred. It is used by OnClickAxis.

TChartClickSeries Type

Unit
TeEngine

Declaration
```
TChartClickSeries = procedure(Sender: TCustomChart; Series: TChartSeries;
ValueIndex: LongInt; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer) of object;
```

Description
The TChartClickSeries type points to a method that notifies a TChart component that a click event has occurred. It is used by the OnClickSeries event.

TChartGradient Component

Properties

Unit
Chart

Description
The TChartGradient component contains all properties used to draw a nice Chart background.

You can change gradient properties by using the TChart.Gradient subcomponent property.

The gradient effect is made of two colors and many middle transition colors between them.

The Visible property controls if the gradient will be displayed or not. Set the Direction property to the desired orientation. Use the StartColor and EndColor properties to change the gradient colors.

The Owner public read-only property refers to the Chart component that owns the gradient.

Warning:
TeeChart do not uses custom color palettes. That means better results can be obtained using 16k or greater video color modes.

Properties

Run-time only

Key Properties

Direction

EndColor

StartColor

Visible

TChartLegend Component

Properties Methods

Unit

TeEngine

Description

TChartLegend a TChart subcomponent.

It draws a rectangle filled with the Chart Series Titles or a Series Values formatted representation. You switch this operation mode by setting the LegendStyle.

The Alignment property defines the Legend position.

Legend can be currently placed at Top, Left, Right and Bottom side of Chart.

Left and Right Legend alignments define a vertical Legend with currently one single column of items.

Top and Bottom Legend alignments define an horizontal Legend with currently one single row of items.

The Legend itself automatically reduces the number of displayed legend items based on the available charting space.

The Legend.ResizeChart property controls if Legend dimensions should be used to reduce Chart points space.

The Legend.OnGetLegendRect event provides a mechanism to supply the desired Rectangle Legend dimensions and placement.

The Legend.OnGetLegendPos event can be used to specify fixed Legend items X Y coordinates.

The Legend.HorizMargin and VertMargin properties control distance between Legend and Left and Right margins.

The Legend.TopPos property can be used in Left or Right Legend alignments to control vertical distance between Legend and Top Chart Margin.

These techniques allow almost complete Legend control.

TChartLegend is quite a big component with many available formatting and dimensioning properties.

Methods

Key Methods

FormattedLegend

FormattedValue

GetColorRect

MaxLegendWidth

TotalLegendItems

Properties

Run-time only

Key Properties

Alignment, Color, ColorWidth, DividingLines, Frame, HorizMargin, Inverted, LegendStyle, ResizeChart, ShadowColor, ShadowSize, TextStyle, FirstValue, Font, ParentChart, RectLegend, TopPos, Visible

TChartListOrder Type

Unit

Declaration

```
TChartListOrder =( loNone, loAscending, loDescending );
```

Description

TChartListOrder defines the possible values of the TChartValueList.Order property.

TChartPen Component

Properties

Unit

TeEngine

Description

TChartPen is an inherited TPen persistent object. Many TeeChart properties are TChartPen properties. For example, the TChartLegend.Frame property is a TChartPen. The Legend draws a frame around it with this Pen settings, but, ONLY when the Visible property is TRUE. Please refer to Delphi TPen help page for more details about TPen objects.

Properties

Run-time only

Key Properties

Color

Mode

Style

Visible

Width

TChartSeries Component

Properties Methods Events

Unit

TeEngine

Description

TChartSeries is the ancestor of every chart series type.

Click here to see the included TeeChart Series types.

Events

Key Events

AfterDrawValues

BeforeDrawValues

OnAfterAdd

OnBeforeAdd

OnClearValues

OnClick

OnGetMarkText

Methods

Key Methods

Add

Delete MaxXValue

AddValue DoSeriesClick MaxYValue

AddNull DrawValuesForward MinXValue

AddXY FillSampleValues MinYValue

AddY GetCursorValueIndex NumSampleValues

AssignValues GetCursorValues PointOrigin

CalcXPos GetEditorClass RefreshSeries

CalcXPosValue GetHorizAxis Repaint

CalcXSizeValue GetMarkValue SameClass

CalcYPos GetVertAxis SwapValueIndex

CalcYPosValue GetYValueList ValuesListCount

CalcYSizeValue IsValidSeriesSource VisibleCount

CalcZOrder IsValidSourceOf XScreenToValue

CheckDataSource MandatoryValueList XValueToText

Clear MarkPercent YScreenToValue

Clicked MaxMarkWidth YValueToText

ColorRange

Count SetFunction

Properties

Run-time only

Key Properties

Active
AllowSinglePoint
ColorEachPoint
ColorSource
Cursor
DataSource
Datasources
DesignMaxPoints
FirstValueIndex
FunctionType
HorizAxis
LastValueIndex
LinkedSeries
Marks
ParentChart
PercentFormat
RecalcOptions
SeriesColor
ShowInlegend
Title
ValueColor
ValueFormat
ValueList
ValueMarkText
VertAxis
XLabel
XLabelsSource
XValue
XValues
YValue
YValues
ZOrder

TChartSeriesList Component

Unit
TeEngine

Ancestor
TList

Description
TChartSeriesList is a non-visible component. Maintains a list of series included in TChart. You can double-click the Chart.Series property to access to an editor dialog.

Using the Chart.Series property is like using a standard array. You can refer to a specific series by number: (Index starts at zero)

Chart1.Series[3].Title:='Boston Sales';

That can be useful when you do not have access to a TChartSeries or when you are doing generic code:

Example:

```
for t:=0 to Chart1.SeriesCount-1 do
  if Chart1.Series[t] is TLineSeries then
     (Chart1.Series[t] as TLineSeries).LinePen.Width:=3;
```

TChartTitle Component

Properties Methods

Unit

Chart

Description

The TChartTitle component is used to display Chart Title and Foot text strings at top and bottom Chart sides respectively. The Text property contains the text to be displayed. This property is of type TStrings which means that it is a series of strings in a list. Whenever you change the Text property you should repaint the Chart component manually. It's not done automatically.

Example:

```
With Chart1.Title.Text do
begin
  Clear;
  Add('ACME Monthly Sales');
  Add('Year: 1997');
end;
Chart1.Repaint;
```

Set the Visible property to True to show the Title or Foot contents. Change the text appearance using the Font property. You can draw a frame around text by setting the Frame pen properties. The frame background color and pattern is defined by the Brush property. The AdjustFrame boolean property controls if title frame will be drawn around title text or using the whole Chart width. Title text can be aligned using the Alignment property.

The ParentChart public read-only property refers to the Chart component that owns the title.

Methods

Key Methods

Clicked

Properties

Run-time only

Key Properties

AdjustFrame

Alignment

Brush

Color

Font

Frame

ParentChart

Text

TitleRect

Visible

TChartValueList Component

Properties Methods

Unit

TeEngine

Description

The TChartValueList component is the container of point values.

Each Series component has one or more values lists.

The XValues and YValues properties are TChartValueList components.

Some other Series components have more values lists. For example, BubbleSeries component has another value list: RadiusValues.

Methods

Key Methods

Last

Locate

Count

Delete

First

Scroll

sort

Properties

Run-time only

Key Properties

DateTime

TempValue

Multiplier
MaxValue
Name
Order
Owner
TotalABS
MinValue
Total
Value
ValueSource

TCanvas3D Component

Methods Properties

Unit

TeCanvas

Description

The TCanvas3D component controls access to TeeChart's 3D Canvas draw properties and methods.

Methods

Run-time only

Key Properties

Calculate3DPosition

ClipRectangle

Arrow

Cylinder

HorizLine3D

VertLine3D

ZLine3D

EllipseWithZ

FrontPlaneBegin

FrontPlaneBegin

FrontPlaneEnd

LineWithZ

LineTo3D

MoveTo3D

Pie3D

Plane3D

PlaneWithZ

PlaneFour3D

Pyramid

RectangleWithZ

RectangleY

RectangleZ

RotateLabel3D

TextOut3D

TriangleWithZ

UnClipRectangle

Properties

Run-time only

Key Properties

Font

Brush

Canvas

Pen

TView3DOptions Component

Properties

Unit
TeCanvas

Description
The TView3dOptions component controls access to overall Chart Zoom and position.

Properties

Run-time only

Key Properties

Elevation

HorizOffset

Orthogonal

Perspective

Rotation

Tilt

VertOffset

Zoom

TChartWall Component

Properties

Unit

Chart

Description

The TChartWall component controls how to display the Chart left and bottom walls.

It is used by the TChart.LeftWall and TChart.BottomWall properties.

The Size property determines the walls dimensions. The Color and Brush properties are used to fill walls, and the Pen property controls the walls frame.

Setting the Color property to "clTeeColor" means walls will not be filled so they will look transparent.

The ParentChart public read-only property refers to the Chart component that owns the wall.

Properties

Run-time only

Key Properties

Brush

Color

ParentChart

Size

Pen

TCircledSeries Component

Properties Methods

Unit
Series

Ancestor
TChartSeries

Description
The TCircledSeries component is an abstract component used as the ancestor of TPieSeries component.

Set the ParentChart property to the desired Chart component.

To add points to the Series you should use the TPieSeries.Add method.

Right-click the component at design-time to access the Chart editor dialog to connect the Series to another Series or to any Table or Query component.

The Circled property determines if the Series will draw as a circle or ellipse.

Change the circle background color using the CircleBackColor property.

The RotationAngle property determines the offset angle used to calculate points positions.

Use the PiePen property to change the circle lines format.

You can define exact circle dimensions by setting the CustomXRadius and CustomYRadius properties.

The Rotate method can be used to change the RotationAngle property at run-time.

The XRadius and YRadius read-only properties return the current circle half-width and half-height in pixels.

The CircleRect read-only property returns the circle bounding rectangle in pixels.

The CircleWidth and CircleHeight read-only properties return the circle width and height in pixels.

The CircleXCenter and CircleYCenter read-only properties return the circle center coordinates in pixels.

When Chart.View3D property is True, the Offset3D read-only property returns the amount in pixels of the circle 3D effect.

Please refer to TChartSeries ancestor description for all common Series properties like Marks, Axis dependence and events.

Methods

Key Methods

AngleToPos

Rotate

Properties

Run-time only

Key Properties

Circled

PiePen

CircleXCenter

CircleYCenter

RotationAngle

XRadius

CustomXRadius

CustomYRadius

YRadius

TCustomBarSeries Component

Properties Methods Events

Unit

Series

Ancestor

TChartSeries

Description

The TCustomBarSeries component is an abstract component used as the ancestor of TBarSeries and THorizBarSeries components.

Set the ParentChart property to the desired Chart component.

To add points to the Series you should use the TBarSeries.AddBar or THorizBarSeries.AddBar methods.

Right-click the component at design-time to access the DataSource Wizard Dialog to connect the Series to another Series or to any Table or Query component.

TCustomBarSeries provides several methods and read-only properties shared by both TBarSeries and THorizBarSeries:

The NormalBarColor,DarkColor and DarkerColor read-only properties return the colors used to fill bar sides and top covers. The CalcBarColors method calculates the above color properties for each bar point.

The DrawBar method is used to display each bar point.

The BarRectangle, BarPolygon and BarEllipse methods can be used to custom draw bar shapes.

The GetBarStyle function returns the corresponding BarStyle for a given bar point.

The BarMargin function returns the amount in pixels used to leave margins from Chart edges.

The ApplyBarOffset function returns a given bar screen coordinate after applying the OffsetPercent property.

Please refer to TChartSeries ancestor description for all common Series properties like Marks, Axis dependence and events.

Events

Key Events

OnGetBarStyle

Methods

Key Methods

ApplyBarOffset

BarMargin

Clicked

GetBarStyle

Properties

Run-time only

Key Properties

BarBrush
BarPen
BarStyle
BarWidthPercent
Dark3D
DarkColor
DarkerColor
MultiBar
NormalBarColor
OffsetPercent
SideMargins
UseYOrigin
YOrigin

TCustomSeries Component

Properties Events

Unit

TCustomSeries is an abstract class derived from TChartSeries.

Its the common ancestor for TLineSeries, TAreaSeries and the TPointSeries.

Events

Key Events

OnClick

Properties

Run-time only

Key Properties

AreaBrush
AreaColor
AreaLinesPen
DrawArea
HorizAxis
LineBrush
LinePen
Pointer
SeriesColor
Stairs
VertAxis

TDBChart Component

Properties Methods Events

Unit

DBChart

Ancestor

TCustomChart, inherits all properties and methods associated with TChart

Description

TDBChart derives from TChart /TCustomChart and inherits all TChart functionality. When a Chart Series is connected to a TDBChart component, TDBChart looks in the Series DataSource property.

If DataSource is a TTable, TQuery, TClientDataset or any valid Delphi DataSet component, TDBChart will automatically retrieve its records preserving all Filters and Ranges. You can also filter which records would be inserted by using the Series OnBeforeAdd event.

TDBChart also accepts Chart Series which are connected to another Chart Series and also Chart Series whose points are being manually added by coding.

The main difference between TChart and TDBChart is that the last one NEEDS the Borland Database Engine to be correctly installed in the target machine, while TChart does not.

The above would be useful in case your application do not need Tables, Querys or any standard Delphi database components.

Changing from a TChart to a TDBChart or vice versa can be done both at design and runtime by changing the Series ParentChart and the Series Values ValueSource properties.

Events

Key Events

OnProcessRecord

Methods

Key Methods

CheckDatasource

IsValidDataSource

RefreshData

RefreshDataSet

Properties

Run-time only

Key Properties

AutoRefresh

RefreshInterval

ShowGlassCursor

TDateTimeStep Type

Unit
TeEngine

Declaration
```
TDateTimeStep = (dtOneSecond, dtFiveSeconds, dtTenSeconds, dtFifteenSeconds,
dtThirtySeconds, dtOneMinute, dtFiveMinutes, dtTenMinutes, dtFifteenMinutes,
dtThirtyMinutes, dtOneHour, dtTwoHours, dtSixHours, dtTwelveHours, dtOneDay,
dtTwoDays, dtThreeDays, dtOneWeek, dtHalfMonth, dtOneMonth, dtTwoMonths,
dtSixMonths, dtOneYear);
```

Description
TDateTimeStep is a set of constants used to specify a date time increment.

Example:
```
Chart1.BottomAxis.Increment := DateTimeStep[ dtTwoDays ];
```
This array is initialized internally at CHART.DCU unit.

Example:
```
DateTimeStep[ dtFifteenMinutes ] := EncodeTime( 0, 15, 0) ;
DateTimeStep[ dtThreeDays ] := 3 ;
```

TDivideTeeFunction component Example

To set the period for TDivideTeeFunction you should use the FunctionType property of TChartSeries

To define a function series by code you should first create a new series for the function. The series may be of any type.

```
{ Set the function using the SetFunction method}.
Series1.SetFunction(TAddTeeFunction.Create(Self));
{You may then define the period for the function - here setting it to to 5}
Series1.FunctionType.Period:=5;
```

To undefine (delete) a function defined for the series use

```
Series1.SetFunction(nil);
```

TDivideTeeFunction Component

See also Example

Unit
TeeFunci

Ancestor
TTeeFunction

Description
TDivideTeeFunction may be added to your project by Chart Editor at design time or at runtime using code. Default period for TDivideTeeFunction is 1 (By default, divide will show the division of all series at each axis point). Period is applicable only to the number of axis points, Period 1 = 1 axis point; period 2 = 2 axis points, etc..

TDivideTeeFunction will work with any number of input series as datasource. The Datasource is best defined using the Chart Editor.

To see a visual representation of TeeChart Standard Functions, go to the TeeChart User Guide.

See Also

TAddTeeFunction

TMultiplyTeeFunction

TSubtractTeeFunction

TAverageTeeFunction

THighTeeFunction

TLowTeeFunction

FunctionType_property

TeeCreateMetafile Example

You should always delete the returned Metafile handle after using it:

```
Var h : HMetafile ;
h := Chart1.TeeCreateMetafile ;
try
....
finally
  DeleteObject( h );
end;
```

TeeCreateMetafile Method

See also Example

Applies to

TChart component

Declaration

```
function TeeCreateMetafile( Enhanced:Boolean; Const Rect:TRect):TMetafile;;
```

Description

The TeeCreateMetafile function returns a new Metafile handle. Metafiles are vector graphic formats. See the Windows API regarding Metafile details. You should always delete the returned Metafile handle after using it:

See Also
TChart.SaveToMetafile

TeeFunci Unit

The TeeFunci unit contains the declarations for the following components and for the enumerated type associated with them. The following items are declared in the this unit:

Components

TAddTeeFunction

TSubractTeeFunction

TMultiplyTeeFunction

TDivideTeeFunction

TAverageTeeFunction

THighTeeFunction

TLowTeeFunction

To see a visual representation of TeeChart Standard Functions, go to the TeeChart User Guide.

Routines

To see a listing of items declared in this unit including their declarations, use the Project Browser.

TFastLineSeries Component

Properties

Unit
Series

Ancestor
TChartSeries

Description
The TFastLineSeries component is an extremely simple Series component that draws its points as fast as possible. To see a visual representation of this Series type, go to the TeeChart User Guide.

It can be used in cases where speed is the most important needed feature. TFastLineSeries has no clicking support and no Marks.

Set the ParentChart property to the desired Chart component.

To add points to the Series use the standard AddY or AddXY methods.

Right-click the component at design-time to access the DataSource Wizard Dialog to connect the Series to another Series or to any Table or Query component.

The only extra property is the LinePen property, which controls the line appearance.

Please refer to TChartSeries ancestor description for all common Series properties like Axis dependence and events.

Properties

Run-time only

Key Properties

HorizAxis

LinePen

SeriesColor

VertAxis

XValues

YValues

TGanttSeries Component

Properties Methods

Unit

GanttCh

Ancestor

TPointSeries

Description

The TGanttSeries component outputs all points as horizontal bars with different starting and ending horizontal coordinates and different vertical positions. To see a visual representation of this Series type, go to the TeeChart User Guide.

It can be used to display a sequence of temporal tasks, sorted over time along the horizontal axis.

Set the ParentChart property to the desired Chart component. Use the AddGantt or AddGanttColor methods to manually fill gantt bar points.

Each gantt bar has an start and end values. These values can be Delphi TDateTime or Double types.

The StartValues property stores all starting values for all Gantt bars. The EndValues property stores all ending values for all Gantt bars.

You can connect gantt bars by using the NextTask property.

Example:

```
Var tmp1,tmp2 : Longint;
{ Clear all Series Gantt bars }
GanttSeries1.Clear;
{ Add a first Gantt bar }
tmp1 := GanttSeries1.AddGantt( EncodeDate( 1996, 1,1 ),
                EncodeDate( 1996, 1,31 ),
                0,
                'Programming' );
{ Add a second Gantt bar }
tmp2 := GanttSeries1.AddGantt( EncodeDate( 1996, 3,1 ),
                EncodeDate( 1996, 3,31 ),
                1,
                'Testing' );
{ Connect the first bar to the second bar }
GanttSeries1.NextTask[ tmp1 ]:= tmp2 ;
```

The ConnectingLinePen property determines the pen used to draw the connecting lines between gantt bar points.

The Pointer.VertSize property specifies the gantt bars height.

Please refer to TPointSeries ancestor description for all common Series properties like Pointer, Marks, Axis dependence, methods and events.

Methods

Key Methods

AddGantt

Properties

Run-time only

Key Properties

ConnectingPen

EndValues

StartValues

NextTask

TGetBarStyleEvent Type

Unit
Series

Declaration
```
TGetBarStyleEvent = procedure( Sender : TCustomBarSeries; ValueIndex :
Longint; Var TheBarStyle : TBarStyle )
```

Description
The TGetBarStyleEvent type points to a method that notifies a TCustomBarSeries component that an event has occurred. It is used by the OnGetBarStyle Event.

TGradientDirection Type

Unit
TeCanvas

Declaration
```
TGradientDirection = ( gdTopBottom, gdBottomTop, gdLeftRight, gdRightLeft );
```

Description
TGradientDirection defines the possible values of the TChartGradient.Direction property.

THighTeeFunction component Example

To set the period for THighTeeFunction you should use the FunctionType property of TChartSeries

To define a function series by code you should first create a new series for the function. The series may be of any type.

```
{ Set the function using the SetFunction method}.
Series1.SetFunction(TAddTeeFunction.Create(Self));
{You may then define the period for the function - here setting it to to 5}
Series1.FunctionType.Period:=5;
```

To undefine (delete) a function defined for the series use

```
Series1.SetFunction(nil);
```

THighTeeFunction Component

See also Example

Unit
TeeFunci

Ancestor
TTeeFunction

Description
THighTeeFunction may be added to your project by Chart Editor at design time or at runtime using code. Default period for THighTeeFunction is 0 (By default, THighTeeFunction will show the highest point of all the series points). Period is applicable to the number of axis points, Period 1 = 1 axis point; period 2 = 2 axis points, etc..

If more than 1 data series is added as datasource for THighTeeFunction then the period default changes to 1.

THighTeeFunction will work with any number of input series as datasource. The Datasource is best defined using the Chart Editor.

To see a visual representation of TeeChart Standard Functions, go to the TeeChart User Guide.

See Also

TAddTeeFunction

TMultiplyTeeFunction

TDivideTeeFunction

TAverageTeeFunction

TSubtractTeeFunction

TLowTeeFunction

FunctionType_property

THorizAxis Type

Unit
TeEngine

Declaration
```
THorizAxis = (aTopAxis,aBottomAxis);
```

Description
THorizAxis defines the possible values of the TChartSeries.HorizAxis property. Most Series components have Horizontal and Vertical dependent axis. You can choose a given Series to have the Horizontal Axis at Top or Bottom.

Default value: aBottomAxis

You can set the HorizAxis property both at design and runtime:

```
LineSeries1.HorizAxis := aTopAxis ;
```

THorizBarSeries Component

Properties Methods

Unit

Series

Ancestor

TCustomBarSeries

Description

The THorizBarSeries component outputs all points as horizontal bars. To see a visual representation of this Series type, go to the TeeChart User Guide.

Several THorizBarSeries can be displayed side-to-side, one behind the other, stacked or stacked 100% by using the MultiBar property.

Set the ParentChart property to the desired Chart component. Use the AddBar or AddXY methods to manually fill area points.

Right-click the component at design-time to access the DataSource Wizard Dialog to connect the Series to another Series or to any Table or Query component.

Warning:

The most important concept in THorizBarSeries is that point values are INVERTED. That means Y values are the bar order position while X values are the bars values.

The AddBar method does the conversion automatically and is the recommended method to add points. When using the AddXY method you should calculate the correct Y position.

Example:

```
{ This code adds 10 horizontal bars.
 The loop counter is used as the Y bar value (the bar order) }
for t:= 1 to 10 do
  Series1.AddXY( MyValues[t], t, MyLabels[t], MyColor[t] );
```

The BarBrush property determines the pattern used to fill bars, while the BarPen property is used to draw the bar edges.

Set the desired bar style (cilynder, pyramid, etc) by changing the BarStyle property.

Use the BarWidthPercent property to control the relative distance between bars. You can specify an exact bar height by using the CustomBarHeight property.

The Dark3D property controls if bar sides are filled with a darker color than front bar faces.

The OffsetPercent property determines the bars vertical displacement. This can be used to create overlayed bars with several bar series components.

Bar series leave margins both at top and bottom Chart sides. You can turn off this default behaviour setting the SideMargins property to False.

By default, bar left positions start at zero horizontal coordinate. Set the YOrigin property to the desired starting left value or set the UseYOrigin to False to make bar left sides start at minimum bars value.

The OnGetBarStyle event can be used to supply a different bar style for each Series point.

Please refer to TCustomBarSeries ancestor description for all common Series properties like Marks, Axis dependence, methods and events.

Methods

Key Methods

AddBar

Clicked
GetOriginPos

Properties

Run-time only

Key Properties

BarHeight

CustomBarHeight

TLegendAlignment Type

Unit
TeEngine

Declaration
```
TLegendAlignment = (laLeft,laRight,laTop,laBottom);
```

Description
TLegendAlignment defines the possible values of the TChartLegend.Alignment property.

TLegendStyle Type

Unit
TeEngine

Declaration
```
TLegendStyle = (lsAuto,lsSeries,lsValues,lsLastValues);
```

Description
TLegendStyle defines the possible values of the TChartLegend.LegendStyle property.

The Chart Legend can draw either the Chart Series Titles or the first Series Values.

This feature is controlled through this property:

- lsAuto — TChartLegend draws Series Titles if there's more than one TChartSeries in the TChart
- lsSeries — TChartLegend draws the Series Titles (also if there's only one TChartSeries in the chart.
- lsValues — TChartLegend draws the first Active TChartSeries values.
- lsLastValues — TChartLegend draws the Last Value of each Active TChartSeries (similar to lsSeries).

TLegendTextStyle Type

Unit
TeEngine

Declaration
```
TLegendTextStyle =
(ltsPlain,ltsLeftValue,ltsRightValue,ltsLeftPercent,ltsRightPercent,ltsXValu
e);
```

Description
TLegendTextStyle defines the possible values of the TChartLegend.TextStyle property.

Several formatting options can be selected when TChartLegend draws the TChartSeries values: You can also delegate the Legend string construction by using the OnGetLegendText Chart event.

ltsPlain	Summer
ltsLeftValue	1234 Summer
ltsRightValue	Summer 1234
ltsLeftPercent	5.1 % Summer
ltsRightPercent	Summer 5.1 %
ltsXValue	4321 (Applies only to Series with X values. See TChartSeries.AddXY method.)

TLineSeries Component

Properties Methods Events

Unit

Series

Ancestor

TCustomSeries

Description

The TLineSeries component outputs all points by drawing a line between them. (To see a visual representation go to the TeeChart User Guide).

Set the ParentChart property to the desired Chart component. Use the AddY or AddXY method to manually fill line points.

Right-click the component at design-time to access the DataSource Wizard Dialog to connect the Series to another Series or to any Table or Query component.

TLineSeries derives from TCustomSeries, which adds support for Series Pointers, Marks and click events.

The LinePen property determines the kind of pen used to draw the lines between points.

Use the LineBrush in 3D mode to change the line fill pattern.

The Stairs property controls how to draw the connecting lines between points. Use then the InvertedStairs property to change how are these steps drawn.

You can show point shapes by using the Pointer subcomponent property.

Please refer to TCustomSeries ancestor description for all common Series properties like Marks, Axis dependence, methods and events.

Events

Key Events

OnClick

Methods

Key Methods

Clicked

Properties

Run-time only

Key Properties

LineBrush

LinePen

InvertedStairs
Stairs
XValues
YValues

TLowTeeFunction component Example

To set the period for TLowTeeFunction you should use the FunctionType property of TChartSeries

To define a function series by code you should first create a new series for the function. The series may be of any type.

```
{ Set the function using the SetFunction method}.
Series1.SetFunction(TAddTeeFunction.Create(Self));
{You may then define the period for the function - here setting it to to 5}
Series1.FunctionType.Period:=5;
```

To undefine (delete) a function defined for the series use

```
Series1.SetFunction(nil);
```

TLowTeeFunction Component

See also Example

Unit

TeeFunci

Ancestor

TTeeFunction

Description

TLowTeeFunction may be added to your project by Chart Editor at design time or at runtime using code. Default period for TLowTeeFunction is 0 (By default, TLowTeeFunction will show the lowest point of all the series points). Period is applicable to the number of axis points, Period 1 = 1 axis point; period 2 = 2 axis points, etc..

If more than 1 data series is added as datasource for TLowTeeFunction then the period default changes to 1.

TLowTeeFunction will work with any number of input series as datasource. The Datasource is best defined using the Chart Editor.

To see a visual representation of TeeChart Standard Functions, go to the TeeChart User Guide.

See Also

TAddTeeFunction

TMultiplyTeeFunction

TDivideTeeFunction

TAverageTeeFunction

THighTeeFunction

TSubtractTeeFunction

FunctionType_property

TMultiArea Type

Unit
Series

Declaration
```
TMultiArea = ( maNone, maStacked, maStacked100 );
```

Description
TMultiArea defines the possible values of the MultiArea property.

TMultiBar Type

Unit
Series

Declaration
```
TMultiBar = (mbNone, mbSide, mbStacked, mbStacked100);
```

Description
TMultiBar defines the possible values of the TCustomBarSeries.MultiBar property.

See Also

TAddTeeFunction

TSubtractTeeFunction

TDivideTeeFunction

TAverageTeeFunction

THighTeeFunction

TLowTeeFunction

FunctionType_property

TMultiplyTeeFunction component Example

To set the period for TMultiplyTeeFunction you should use the FunctionType property of TChartSeries

To define a function series by code you should first create a new series for the function. The series may be of any type.

```
{ Set the function using the SetFunction method}.
Series1.SetFunction(TAddTeeFunction.Create(Self));
{You may then define the period for the function - here setting it to to 5}
Series1.FunctionType.Period:=5;
```

To undefine (delete) a function defined for the series use

```
Series1.SetFunction(nil);
```

TMultiplyTeeFunction Component

See also Example

Unit
TeeFunci

Ancestor
TTeeFunction

Description
TMultiplyTeeFunction may be added to your project by Chart Editor at design time or at runtime using code. Default period for TMultiplyTeeFunction is 1 (By default, multiply will show the multiplication of all series at each axis point). Period is applicable only to the number of axis points, Period 1 = 1 axis point; period 2 = 2 axis points, etc..

Multiply will work with any number of input series as datasource. The Datasource is best defined using the Chart Editor.

To see a visual representation of TeeChart Standard Functions, go to the TeeChart User Guide.

TOnGetLegendPos Type

Unit
TeEngine

Declaration
```
TOnGetLegendPos = Procedure( Sender : TCustomChart; Index : Longint; Var
X,Y,XColor : Longint) of object;
```

Description
The TOnGetLegendPos type points to a method that notifies a TChart component that an event has occurred. It is used by the OnGetLegendPos Event

TOnGetLegendRect Type

Unit
TeEngine

Declaration
```
TOnGetLegendRect = Procedure( Sender:TCustomChart; Var Rect:TRect) of
object;
```

Description
The TOnGetLegendRect type points to a method that notifies a TChart component that an event has occurred. It is used by the OnGetLegendRect Event.

TOnGetLegendText Type

Unit
TeEngin

Declaration
```
TOnGetLegendText = procedure(Sender:TCustomChart; LegendStyle:TLegendStyle;
Index:Longint; Var LegendText:String);
```

Description
The TOnGetLegendText type points to a method that notifies a TChart component that an event has occurred. It is used by the OnGetLegendText event.

Total property Example

This code informs the user the Total sum of “Sales1997” Series values:

```
ShowMessage(‘ Total Sales are: ‘ + FloatToStr( Sales1997.YValues.Total ) );
```

Total Property

See also Example

Applies to

TChartValueListcomponent

Declaration

```
property Total : Double;
```

Description

The Total double property maintains the sum of all TChartValueList values. When adding, deleting or modifying point values using the right methods, Total is automatically incremented and decremented.

Total property is used by some Functions to improve speed when performing calculations against point values, where having already calculated the sum of point values is necessary.

See Also

TChartValueList.TotalABS

TChartValueList.MinValue

TChartValueList.MaxValue

TotalLegendItems Example

This code shows the number of visible Legend items:

```
ShowMessage( IntToStr( TChartLegend(Chart2.Legend).TotalLegendItems ) );
```

Note: Casting to TChartLegend should be done due to Delphi 1 compatibility reasons.

TotalLegendItems Method

Applies to
TChartLegend component

Declaration
```
function TotalLegendItems:Longint;
```

Description
The TotalLegendItems method returns the number of displayed Legend items. Legend.LegendStyle controls to calculate the number of visible items using either all Series Titles or foremost Series point values. It uses FirstValue Legend property to start counting from the desired first Legend item.

See Also

TChartLegend.LegendStyle

TChartLegend.FirstValue

TPanningMode Type

Unit
Chart

Declaration
```
TPanningMode = (pmNone,pmHorizontal,pmVertical,pmBoth);
```

Description
TPanningMode defines the possible values of the AllowPanning property.

pmNone	Deny scrolling.
pmHorizontal	Allow only Horizontal Scrolling.
pmVertical	Allow only Vertical Scrolling.
pmBoth	Allow complete Horizontal and Vertical Scrolling.

TTFunction Component

Properties

Unit
StatChar

Ancestor

Description
TTFunction is an abstract Series type.

While DataSource points are being inserted, TTFunction will trigger a calculation request when the last inserted point has overflowed the specified period.

TAverageTeeFunction is a TTeeFunction descendant.

Properties

- **Run-time only**
- **Key Properties**
  - Period

Period Property

See also Example

Applies to

TTeeFunction component

Declaration

```
property Period : Double;
```

Description

The Period property controls how many points or X range(sse note below) will trigger a new point calculation.

For example, TAverageTeeFunction uses the Period property to calculate a new average point each time the "Period" number of points or X range(see note below) is exceed.

You may switch between number of points or X range by using the TTeeFunction.PeriodStyle property.

See Also
FunctionType_property

Period property Example

For example, this code uses a TTeeFunction to show one point for every 5 points in the Series2 datasource series:

```
Series1.Pointer.Visible:= True;
Series2.Pointer.Visible:= True;
Series1.DataSource:= Series2;
Series1.FunctionType.Period:= 5;
```

TPieSeries Component

Properties Methods

Unit

Series

Ancestor

TCircledSeries

Description

The TPieSeries component outputs all points drawing slices forming an ellipse. To see a visual representation of this Series type, go to the TeeChart User Guide.

Set the ParentChart property to the desired Chart component. Use the AddPie method to manually fill pie slices.

Right-click the component at design-time to access the DataSource Wizard Dialog to connect the Series to another Series or to any Table or Query component.

The PiePen property determine the kind of pen used to draw the slice frames.

The TChart.View3D and TChart.Chart3DPercent properties control the 3D pie appearance.

Set the UsePatterns property to True to fill the pie slices using brush patterns.

The Color3D property controls if the 3D pie region will be colored or not. The ShadowColor and Shadowed3D properties apply a shadow effect on pie 3D region.

The PieValues property can be used to access the pie values.

Please refer to TCircledSeries ancestor description for all common Series properties like Marks, Axis dependence, methods and events.

Methods

Key Methods

AddPie

AngleToPos
CalcClickedPie
Clicked
PointToAngle

Properties

Run-time only

Key Properties

Color3d

Dark3D

ExplodeBiggest

PiePen

PieValues

OtherSlice

ShadowColor

Shadowed3d

UsePatterns

TPointSeries Component

Properties Methods Events

Unit

Series

Ancestor

TCustomSeries

Description

The TPointSeries component outputs all points using the Pointer subcomponent properties. To see a visual representation of this Series type, go to the TeeChart User Guide.

Set the ParentChart property to the desired Chart component. Use the AddY or AddXY method to manually fill point values.

Right-click the component at design-time to access the DataSource Wizard Dialog to connect the Series to another Series or to any Table or Query component.

TPointSeries derives from TCustomSeries, which adds support for Series Pointers, Marks and click events.

The Pointer.Style property determines the kind of shape used to draw the points.

Use the Pointer.Brush property to change the point filling pattern. The Pointer.Pen property defines the kind of pen used to draw the point frames.

Point dimensions can be changed using the Pointer.HorizSize and Pointer.VertSize properties.

Please refer to TCustomSeries ancestor description for all common Series properties like Marks, Axis dependence, methods and events.

Events

Key Events

OnClickPointer

Properties

Run-time only

Key Properties

Pointer

XValues

YValues

TProcessRecordEvent Type

Unit
TeEngine

Declaration
```
TProcessRecordEvent = Procedure( Sender:TDBChart; DataSet:TDataSet) of
object;
```

Description
The TProcessRecordEvent type points to a method that notifies a TDBChart component that an event has occurred. It is used by the OnProcessRecord Event.

TQRChart Component

Properties

Unit

QRTee

Ancestor

The TQRChart component derives from QuickReports TQRPrintable component.

Description

QuickReports is a native Delphi reporting tool Copyright by QuSoft, Norway. It's included with Borland Delphi.

You can contact QuSoft at www.qusoft.no

The TQRChart component links TChart or TDBChart components with QuickReports.

It's an interface between both environments to allow Charts to be embedded in report bands.

The TeePrintMethod property allows two operation modes: The qtmBitmap mode outputs Charts as bitmap pictures. This makes the Chart appear as it looks on the screen, but remember that bitmaps are larger and slower to print.

You can set TeePrintMethod property to qtmMetafile. This is much faster and gives much better output quality than bitmaps. Chart contents are sent to QuickReports in Windows metafile vector instructions. There are, however, some limitations and bugs in the Windows metafile subsystem.

Clipping does not work fine as it uses physical pixel coordinates instead of logical. Rotated fonts dimensions are very difficult to calculate in metafile mode. This makes rotated fonts overlap or leave too much space on axis and titles.

32bit Windows enhanced metafile (EMF) mode promises better results. These problems will be addressed in future TeeChart versions.

Please refer to QuickReports documentation and help file to know all details about creating reports. See the TeeChart User Guide for more information about using TQRChart.

Properties

- **Run-time only**
- **Key Properties**
  - Chart
    - TeePrintMethod

TSeriesClick Type

Unit
TeEngine

Declaration
```
TSeriesClick = procedure(Sender:TChartSeries; ValueIndex: LongInt; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
```

Description
The TSeriesClick type points to a method that notifies a TChartSeries component that an event has occurred. It is used by the OnClick Event.

TSeriesClickPointerEvent Type

Unit
Series

Declaration
```
TSeriesClickPointerEvent = procedure(Sender: TCustomSeries; ValueIndex:
LongInt; X, Y: Integer) of object;
```

Description
The TSeriesClickPointerEvent type points to a method that notifies a TChartSeries component that an event has occurred. It is used by the OnClickPointer Event.

TSeriesMarkPosition Component

Properties Methods

Unit
TeEngine

Description
Component defines position for the Mark referenced by the position index.

Methods

- **Run-time only**
- **Key Properties**

Bounds

Properties

Run-time only

Key Properties

ArrowTo

ArrowFrom

Custom

LeftTop

Height

Width

TSeriesMarksPositions Component

Properties Methods

Unit
TeEngine

Description
Each Series Mark position can be reached and modified via this component.

Methods

Run-time only

Key Properties

Automatic

Properties

Run-time only

Key Properties

Position

TSeriesMarks Component

Properties Methods

Unit

TeEngine

Description

Each Series have a Marks property. The Marks are like Delphi hints for each Series point. Marks can be Visible or not and have many custom formatting properties.

Methods

Run-time only

Key Properties

Clicked

Properties

Run-time only

Key Properties

Arrow
ArrowLength
BackColor
Clip
Font
Positions
Frame
ParentSeries
Style
Transparent
Visible

TSeriesMarksStyle Type

Unit
TeEngine

Declaration
```
TSeriesMarksStyle = (smsValue, smsPercent, smsLabel, smsLabelPercent,
smsLabelValue, smsLegend, smsPercentTotal, smsLabelPercentTotal, smsXValue);
```

Description
TSeriesMarksStyle defines the possible values of the TSeriesMarks.Style property. Series Marks will draw a textual representation of each point values.

smsValue	1234
smsPercent	12 %
smsLabel	Cars
smsLabelPercent	Cars 12 %
smsLabelValue	Cars 1234
smsLegend	(Depends on LegendTextStyle)
smsPercentTotal	12 % of 1234
smsLabelPercentTotal	Cars 12 % of 1234
smsXValue	{ 21/6/1996 }

You can choose both at design and runtime how this text will be constructed: LineSeries1.Marks.Style := smsPercent ;

TSeriesOnAfterAdd Type

Unit
TeEngine

Declaration
```
TSeriesOnAfterAdd = Procedure(Sender:TChartSeries; ValueIndex:Longint) of
object;
```

Description
The TSeriesOnAfterAdd type points to a method that notifies a TChartSeries component that an event has occurred. It is used by the OnAfterAdd Event.

TSeriesOnBeforeAdd Type

Unit
TeEngine

Declaration
```
TSeriesOnBeforeAdd = Function(Sender: TChartSeries): Boolean of object;
```

Description
The TSeriesOnBeforeAdd type points to a method that notifies a TChartSeries component that an event has occurred. It is used by the OnBeforeAdd event.

TSeriesOnClear Type

Unit
TeEngine

Declaration
```
TSeriesOnClear = Procedure(Sender: TChartSeries) of object;
```

Description
The TSeriesOnClear type points to a method that notifies a TChartSeries component that an event has occurred. It is used by the OnClearValues Event.

TSeriesOnGetMarkText Type

Unit
TeEngine

Declaration
```
TSeriesOnGetMarkText = procedure ( Sender : TChartSeries ; ValueIndex :
Longint ; Var MarkText : String )
```

Description
The TSeriesOnGetMarkText type points to a method that notifies a TChartSeries component that an event has occurred. It is used by the OnGetMarkText Event.

TSeriesPointer Component

Properties Methods

Unit

Series

Description

Some Series have a Pointer property. Pointers are shape figures drawn on each Y point coordinates. You can set several properties to change the pointers appearance.

Methods

Key Methods

ChangeHorizSize

ChangeStyle

ChangeVertSize

Draw

DrawPointer

PrepareCanvas

Properties

Run-time only

Key Properties

Brush

Draw3d

HorizSize

InflateMargins

ParentSeries

Pen

Style

VertSize

Visible

TSeriesPointerStyle Type

Unit
Series

Declaration
```
TSeriesPointerStyle = (psRectangle, psCircle, psTriangle, psDownTriangle,
psCross, psDiagCross, psStar, psDiamond, psSmallDot);
```

Description
TSeriesPointerStyle defines the possible values of the TSeriesPointer.Style property. This is the shape a Pointer has.

psRectangle

psCircle

psTriangle

psDownTriangle

psCross

psDiagCross

psStar

You can change the pointer Style both at design and runtime: PointSeries1.Pointer.Style := psTriangle ;

TSeriesRecalcOptions Type

Unit
TeEngine

Declaration
```
TSeriesRecalcOptions = set of (rOnDelete, rOnModify, rOnInsert, rOnClear);
```

Description
TSeriesRecalcOptions defines the possible values of the RecalcOptions property.

TSubtractTeeFunction component Example

To set the period for TSubtractTeeFunction you should use the FunctionType property of TChartSeries

To define a function series by code you should first create a new series for the function. The series may be of any type.

```
{ Set the function using the SetFunction method}.
Series1.SetFunction(TAddTeeFunction.Create(Self));
{You may then define the period for the function - here setting it to to 5}
Series1.FunctionType.Period:=5;
```

To undefine (delete) a function defined for the series use

```
Series1.SetFunction(nil);
```

TSubtractTeeFunction Component

See also Example

Unit
TeeFunci

Ancestor
TTeeFunction

Description
TSubtractTeeFunction may be added to your project by Chart Editor at design time or at runtime using code. Default period for TSubtractTeeFunction is 1 (By default, subtract will show the subtraction of one series from another at each axis point). Period is applicable only to the number of axis points, Period 1 = 1 axis point; period 2 = 2 axis points, etc..

Subtract will only work with 2 input series. The first series in the Chart Editor list will subtract the 2nd series.

To see a visual representation of TeeChart Standard Functions, go to the TeeChart User Guide.

See Also

TAddTeeFunction

TMultiplyTeeFunction

TDivideTeeFunction

TAverageTeeFunction

THighTeeFunction

TLowTeeFunction

FunctionType_property

TTeeBackImageMode Type

See also

Unit
Chart

Declaration
```
TTeeBackImageMode = ( pbmStretch, pbmTile, pbmCenter);
```

Description
TTeeBackImageMode defines the possible values of the BackImageMode property.

pbmStretch : The BackImage will be resized to fit Chart dimensions.

pbmTile : The BackImage will be tiled.

pbmCenter : The BackImage will not be resized.

See Also

TChart.BackImage

TChart.BackImageInside

TChart.BackImageMode

TTeeFunction Component

Properties

Unit
StatChar

Ancestor
TComponent

Description
TTeeFunction is an abstract Series type.

While DataSource points are being inserted, TTeeFunction will trigger a calculation request when the last inserted point has overflowed the specified period.

TAverageTeeFunction is a TTeeFunction descendant.

Properties

- **Run-time only**
- **Key Properties**
  - Period

TTeeQuickMethod Type

Unit
TeEngine

Declaration
```
TTeeQuickMethod = ( qtmBitmap, qtmMetafile );
```

Description
TTeeQuickMethod defines the possible values of the TeePrintMethod property.

TValueEvent Type

Unit
TeEngine

Declaration
```
TValueEvent = (veClear, veAdd, veDelete, veRefresh, veModify);
```

Description
Series notify the following events to their dependent Series:

veClear The Series have been emptied.

veAdd A new point has been appended.

veDelete A point has beed deleted.

veRefresh Series need to be refreshed (reconstructed).

This events are intended for Series developers.

TVertAxis Type

Unit
TeEngine

Declaration
```
TVertAxis = (aLeftAxis,aRightAxis);
```

Description
TVertAxis defines the possible values of the TChartSeries.VertAxis property. Most Series components have Horizontal and Vertical dependent axis. You can choose a given Series to have the Vertical Axis at Left or Right.

Default value: aLeftAxis

You can set the VertAxis property both at design and runtime:

```
LineSeries1.VertAxis := aRightAxis ;
```

TeEngine Unit

The TeEngin unit contains the declarations for the following components and for the enumerated type associated with them. When you add a component declared in this unit to a form, the unit is automatically added to the uses clause of that form's unit. The following items are declared in the this unit:

Components

TChartAxis

TChartAxisTitle

TChartLegend

TChartPen

TChartSeries

TChartSeriesList

TChartTitle

TChartValueList

TSeriesMarks

Types

TAxisLabelStyle

TAxisOnGetLabel

TDateTimeStep

TeeDefaultCapacity

THorizAxis

TLegendStyle

TLegendTextStyle

TOnGetLegendText

TSeriesClick

TSeriesMarksStyle

TSeriesOnGetMarkText

TSeriesRecalcOptions

TValueEvent

TVertAxis

Routines

DaysInMonth

To see a listing of items declared in this unit including their declarations, use the Project Browser.

Component Reference

Quick Start About.. Contact us..

The main TeeChart component is: TChart.

TChart is a placeholder for TChartSeries components. TChart is derived from TPanel component, and inherits all its functionality.

When you place a TChart component in a Delphi Form, you will see a standard TPanel with an upper title and a lower foot text.

Right click on the Chart to see the opening TeeChart menu where you can select the Chart editor alternatively double-click to directly call the Chart editor.

The Chart editor presents Chart configuration options and the TeeChart Gallery which contains access to all data series types. Data series types are subcomponents that can be accessed and manipulated independently of the Chart and are accessible across Charts.

For Database charts, you need to use: TDBChart

Place one or more series in the Chart and either connect to a DataSource or write your own code to populate the series.

Key palette components

 TChart

 TDBChart

 **TQRChart**

TeeAxisClickGap Global Variable

Unit

TeeProcs

Applies to
Global

Declaration
```
property TeeAxisClickGap : Longint;
```

Description
Maximum number of pixels from the Axis to the mouse cursor to consider the Axis clicked.

TeeChart Wizard

To create a Chart using TeeChart Wizard select the **File** menu in Delphi and **New**. Choose **Business** from the tab selector in the **New Items** window. You will see the TeeChart Wizard icon.

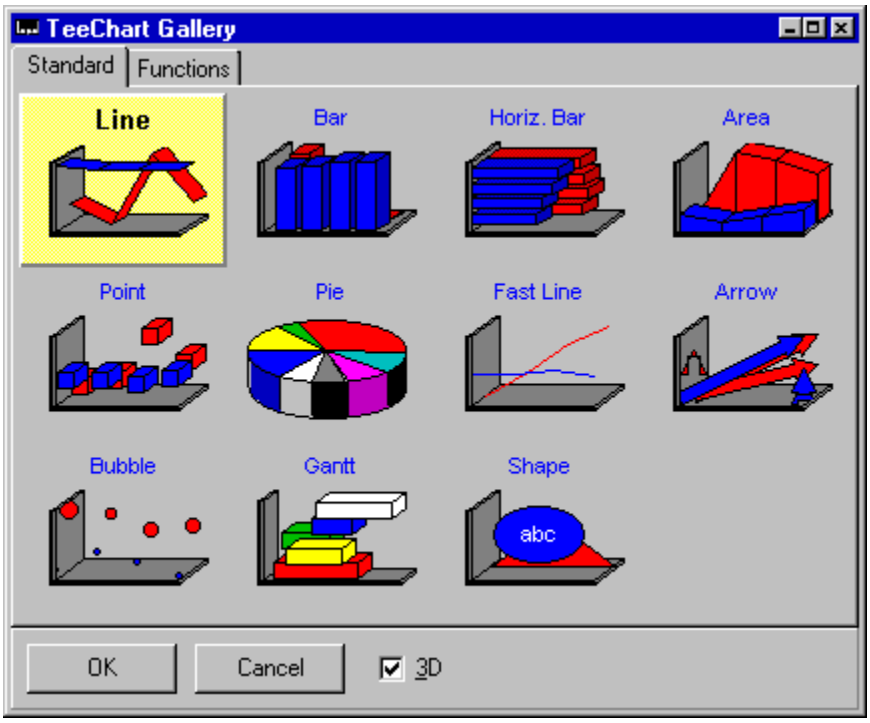
Simply double-click the icon to start creating charts! The Wizard will lead you through the steps necessary to create your own non-database or database aware chart.

TeeChart Gallery

See also

Description

The TeeChart Gallery contains a visual catalogue of all available Series types. You may access the Gallery via the Chart Editor.



See Also
Series Unit

TeeChart Global Constants and Methods

Global Procedures

```
Procedure SwapLongInt(Var a, b : LongInt);
Procedure SwapDouble(Var a, b : Double);
```

These procedures interchange the "a" and "b" parameter values. You can use it in your applications.

```
Procedure TeeSplitInLines(Var St : String; Separator : String);
```

This procedure can be used to split Labels to Multiline replacing the Separator string with a carriage return..

Gradient Fill

```
procedure GradientFill(Canvas: TCanvas; Const Rect: TRect; top, bottom:
TColor; Horizontal: Boolean);
```

The GradientFill method is used to fill a Screen area with multi-colored lines to obtain a nice shadow effect and coloured backgrounds. The Chart.Gradient component uses this method internally.

Global Functions

DaysInMonth

```
Function DaysInMonth(Year,Month);
```

This function returns the number of days that a given pair of Year and Month has. It is declared in TeEngine

ApplyDark

```
Function ApplyDark(Color: TColor; HowMuch: Byte) : TColor;
```

This function returns a darker color than the Color parameter. The RGB values of Color parameter are increased "HowMuch" units.

```
Function GetDefaultColor(t: Longint): TColor;
```

Returns the t th color in the default array palette.

```
Function MaxLong(a,b:Longint):Longint; returns max (> a , b)
Function MinLong(a,b:Longint):Longint; returns min (< a , b)
```

TeeChart Global Constants:

DateTimeStep

A global array of constants useful to define DateTime periods.

```
ChartMarkColor : Longint = $80FFFF
```

This is the default color for Series Marks rectangles. (Yellow)

```
PercentOf : String[10] = ' of '
```

Used to show " 35 % of 1234 " in Series Marks.

TeeChart Pro

TeeChart Pro order information License and disclaimer

TeeChart Pro is a comprehensive charting tool aimed at those developers wishing to program with or make use of extended TeeChart functionality. **TeeChart Pro** includes the option to purchase **100% source code.**

For an overview of TeeChart functionality available with Delphi version 5 see TeeChart version 4

Select this link to see What's new in TeeChart 4 Pro and Standard versions.

Features of **TeeChart Pro**:

- 100 % Source code
- 16-bit version for Delphi 1
- 32-bit version for Delphi 2, 3, 4 and 5 and Borland C++ Builder 1, 3 and 4
- 11 standard Series types
- 9 Extended Series types
- 6 Custom Sample Series types
- 16 Statistical functions
- Data aware
- Series Gallery
- 2D, 3D plus OpenGL 3D
- QuickReport Integration
- Zoom, scroll and real time
- Royalty free
- Custom drawing
- Custom printing
- Extensive demos
- Design time integrated Chart and Series editor
- Online help
- Electronic reference manual (Word format)
- Custom Series build
- Developer Custom Series guide
- Sample Custom Series and Functions
- Extended Statistical functions
- Extensive demo code
- Runtime Chart Editor
- Extended Online help
- Printed reference guide and electronic developer guide

Programming environments

TeeChart Pro order information TeeChart Pro

TeeChart is written as a 100% Delphi native VCL. It supports Borland's 16 bit (Delphi 1) and 32 bit (Delphi 2, 3, 4 and 5 and C++ Builder v1, 3 and 4) programming environments.

TeeChart version 4

License and disclaimer TeeChart Pro

TeeChart 4 runtime version is packaged with **Delphi version 5**. It is a fully functional charting library containing a subset of the powerful charting capabilities of TeeChart Pro.

Features of TeeChart 4 runtime version:

- 32-bit version for Delphi 3, 4 and 5
- 11 standard Series types
- Statistical functions
- Data aware
- Series Gallery
- 2D and 3D
- QuickReport Integration
- Zoom, scroll and real time
- Royalty free
- Custom drawing
- Custom printing
- Extensive demos
- Design time integrated Chart and Series editor
- Online help including user guide

TeeChart Pro

See TeeChart Pro for information about extended features and availability of 100% TeeChart source code.

We hope you'll enjoy using TeeChart !!!

Thanks to All of You !

Ordering TeeChart Pro

If you would like to order TeeChart Pro you can find an automated order form in the opening menu when right mouse clicking on a TChart in Delphi. It will create a custom order form for you. Alternatively use the form included below. Please contact us if you have any questions.

TeeMach SL

Rocafort 35-37 5o3a

08015 Barcelona, Catalonia

Spain

****

TeeChart - Pro v4.0 ORDER FORM

****

WWW: http://www.teechart.com

Please fill in the address information below. If you require TeeChart Pro to be mailed to you this address will be used for mailing purposes:

Customer Information:

Name..…… :

Company…… :

Address..…… :

City..…… :

State.…….. :

ZIP...…….. :

Country…….. :

Phone.…… :

FAX...……. :

EMail.……. :

(European Community only)

VAT or company reference number ..:

Credit Card Information:

Credit Card Type…...…… :

Credit CardHolder Name…... :

Credit Card Number...…… :

Credit Card Exp. Date…… : /

Order Product:

TeeChart Pro v4 is available **with** or **without source code**. Please mark your choice in the tables below:

Prices WITHOUT source code:

Product Code	Description	Price/license	Quantity	Total (US$)

401	Upgrade from TeeChart v1.03/v3	US$39
40	Single License	US$139
421	Site license	Please consult teeMach for details

Prices WITH 100% TeeChart Pro source code:

Product Code	Description	Price/license	Quantity	Total (US$)
402	Upgrade from TeeChart v1.03/v3 with source code	US$70		
41	Single License with source code	US$279		
422	Site license with source code	Please consult teeMach for details		

Shipping & Handling:

(For those not requiring a printed introductory guide we are able to make TeeChart Pro

available electronically to avoid shipping charges). The introductory guide includes information about help and information sources for TeeChart Pro and is included in Word format with the electronic only product.

I Would like printed introductory guide ☐ (Check box to receive)

(Single license software pack includes 1 disk and 1 introductory guide)

Region	Post	(add per additional license)
Europe	US$15	(US$10)
USA	US$25	(US$10)
Other	US$30	(US$10)

(Any questions about non-standard or courier shipment - please contact us)

Shipping :US$

T O T A L :US$

We accept cheques (eurocheques in pesetas please) or cash on delivery. For purchases by bank transfer please contact us for information.

FAX this Order Form to us at FAX NUMBER: +34 972 59 71 75

Or...

Send by Internet EMail to: sales@teemach.com

Or...

Send by postal mail to:

teeMach SL.

Rocafort 35-37 5o3a

08015 Barcelona, Catalonia

Spain

If you have ordered TeeChart-Pro to be sent to you it will be mailed to you by urgent post.

You should expect to receive it in few days.

Your email address is important for confirmation information should you wish to download TeeChart-Pro from the website.

TeeChart Quick Start

Please read the topics below they will guide you through basic steps of chart creation.

After you have installed the TeeChart components, open the TEEDEMO.DPR Demo project or start a new Form and place a Chart on it.

Create a new Data-Aware Chart and get values from a Table or Query

Create a new Chart and insert values MANUALLY

Learn more about TeeChart components

TeeChart license and disclaimer

TeeChart Pro About

License

TeeChart 4 runtime version is included with Delphi version 4 and 5. No additional license is required to use this software. Please check with Borland documentation for details of your license agreement.

Disclaimer

The Author cannot and does not guarantee that any functions contained in the Software will meet your requirements, or that its operations will be error free. The entire risk as to the Software performance or quality, or both, is solely with the user and not the Author. You assume responsibility for the selection of the component to achieve your intended results, and for the installation, use, and results obtained from the Software.

The Author makes no warranty, either implied or expressed, including without limitation any warranty with respect to this Software documented here, its quality, performance, or fitness for a particular purpose. In no event shall the Author be liable to you for damages, whether direct or indirect, incidental, special, or consequential arising out the use of or any defect in the Software, even if the Author has been advised of the possibility of such damages, or for any claim by any other party.

All other warranties of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, are expressly excluded.

TeeChart Series Types

TeeChart Gallery

TeeChart Library includes the most typical Chart Series types.

Each type has its own specific properties and events to customize your Charts.

Most importantly you can derive new subcomponents from any Series type, thus creating any NEW Chart types.

TeeChart includes the following Chart Series types (To see a visual representation of these Series go to the TeeChart User Guide):

Standard:

Line

Area

Point (scatter)

Bar,Pyramid,Cilinder...

Horizontal Bar,Pyramid,Cilinder...

Pie

Shapes

Fast Line

Arrows

Gantt

Bubble

All Series are derived from the "pseudo-abstract" TChartSeries class.

TChartSeries hides all the logic complexity leaving the developer to code the "drawing" and "aesthetic" parts of a given Series type.

See TTeeFunction for an outline description of available statistical functions.

Some advanced Series features include custom scaling and margining. Other "pseudo-abstract" components are available for component developers:

Period

TeeClipWhenMetafiling Global Variable

See also

Unit

TeeProcs

Applies to

Global

Declaration

```
property TeeClipWhenMetafiling : Boolean;
```

Description

This property forces clipping when creating metafile charts.

See Also
TeeClipWhenPrinting

TeeClipWhenPrinting Global Variable

See also

Unit

TeeProcs

Applies to
Global

Declaration

```
property TeeClipWhenPrinting : Boolean;
```

Description
This property forces clipping when printing charts.

See Also
TeeClipWhenMetafiling

TeeDefaultCapacity Global Variable

See also

Applies to
Global

Declaration
```
property TeeDefaultCapacity : Longint;
```

Description
Default 0

This global variable is the default capacity of all internal TList objects. See the Delphi help file regarding TList.Capacity property.

Setting it to a bigger number can increase speed when adding many points to Series components. TeeChart always uses this variable to initialize the TList.Capacity of all internal Lists.

See Also
TChartValueList

TeeEraseBack Global Variable

See also

Applies to
Global

Declaration
```
property TeeEraseBack : Boolean;
```

Description
Default True

The TeeEraseBack global variable determines if TeeChart will handle the Windows WM_ERASEBACK message. This messages forces TChart to fill the panel background before repainting.

Setting it to False prevents TeeChart to erase the panel background thus avoiding flickering when resizing Charts in Windows when full-window-drag is selected.

It affects all Chart components as it's a global variable.

See Also
TChart.BufferedDisplay

TeePrintMethod Property

See also

Applies to

TQRChart component

Declaration

```
property TeePrintMethod : TTeeQuickMethod;
```

Description

Default qtmMetafile

The TeePrintMethod property indicates the method used to display an embedded TChart or TDBChart component into a QuickReport Band component. When using the qtmMetafile mode, all drawing instructions are generated directly over the QuickReport Band. When using qtmBitmap, the whole Chart is outputted to an internal Bitmap and then copied to the QuickReport band. The qtmMetafile method guaranties better Printing and Screen resolution and it's faster in most situations.

See Also

TChart.BufferedDisplay

TChart.TeeCreateMetafile

TChart.SaveToBitmapFile

TeeScrollKeyShift Global Variable

See also

Unit

Chart

Applies to
All TeeChart components

Declaration
```
property TeeScrollKeyShift : TShiftState;
```

Description
Default Value: []

Keys that should be pressed to start scroll.

Warning: This is not a TChart property. It's a global variable.
For Windows users with swapped mouse buttons, this variable points to the opposite mouse button.

See also

TeeZoomMouseButton

TeeZoomKeyShift

TChart.OnScroll

TeeScrollMouseButton Global Variable

See also

Unit

Chart

Applies to
All TeeChart components

Declaration

```
property TeeScrollMouseButton : TMouseButton;
```

Description
Default Value: mbRight

The TeeScrollMouseButton global variable determines which of the mouse buttons will be used by TChart users at run-time to scroll chart contents.

By default it is assigned to the Right mouse button. You can change this behavior setting it to a different mouse button.

The other TeeZoomMouseButton global variable determines the mouse button used to zoom chart contents.

Never set the same mouse button for these two variables.

Warning: This is not a TChart property. It's a global variable.
For Windows users with swapped mouse buttons, this variable points to the opposite mouse button.

See also

TeeZoomMouseButton

TChart.OnScroll

TeeZoomKeyShift Global Variable

See also

Unit

Chart

Applies to
All TeeChart components

Declaration

```
property TeeZoomMouseButton : TShiftState;
```

Description
Default Value: []

Key that should be pressed to start zoom. This variable allows control of which mousebutton and keyboard combination should be applied to start zoom.

Warning: This is not a TChart property. It's a global variable.
For Windows users with swapped mouse buttons, this variable points to the opposite mouse button.

See Also

TChart.OnZoom

TChart.UndoZoom

TeeScrollMouseButton

TeeScrollKeyShift

TeeZoomMouseButton Global Variable

See also

Unit

Chart

Applies to
All TeeChart components

Declaration

```
property TeeZoomMouseButton : TMouseButton;
```

Description
Default Value: mbLeft

The TeeZoomMouseButton global variable determines which of the mouse buttons will be used by TChart users at run-time to apply zoom.

By default it is assigned to the Left mouse button. You can change this behavior setting it to a different mouse button.

The other TeeScrollMouseButton global variable determines the mouse button used to scroll chart contents.

Never set the same mouse button for these two variables.

Warning: This is not a TChart property. It's a global variable.
For Windows users with swapped mouse buttons, this variable points to the opposite mouse button.

See Also

TChart.OnZoom

TChart.UndoZoom

TeeScrollMouseButton

TempValue Property

Applies to
TChartValueList component

Declaration
```
property TempValue : Double;
```

Description
Run-time only.

The TempValue property is a special internal property used when adding values from one Series to another or when adding values to a Series with more than "X" and "Y" values, such as TBubbleSeries which has "Radius" values as well.

Text Property

See also Example

Applies to

TChartTitle component

Declaration

```
property Text : TStrings;
```

Description

The Text property determines the text that appears at a Chart Title or Foot. You can add or change text using the standard Delphi TStrings object methods:

Text property Example

You can add or change text using the standard Delphi TStrings object methods:

```
Chart1.Title.Clear;
Chart1.Title.Add('Hello');
Chart1.Title.Add('World');
```

See Also
TChart.Title

TextStyle Property

See also Example

Applies to

TChartLegend component

Declaration

```
property TextStyle : TLegendTextStyle;
```

Description

Default Value: ltsLeftValue

The TextStyle property indicates how Legend text items will be formatted.

ltsPlain shows the point Label only.

ltsLeftValue shows the point Value and the point Label.

ltsRightValue shows the point Label and the point Value.

ltsLeftPercent shows the percent the point represents and the point Label.

ltsRightPercent shows the point Label and the percent the points represent.

ltsXValue shows the point's X value. It applies only to Series with X (horizontal) values.

Values are pre-formatted using the Series ValueFormat property. Percents are pre-formatted using the Series PercentFormat property.

You can also use the TChart.OnGetLegendText event to supply customized Legend texts.

TextStyle property Example

These are examples of different Legend's TextStyle values:

```
ltsPlain      Summer
ltsLeftValue    1234 Summer
ltsRightValue    Summer 1234
ltsLeftPercent   5.1 % Summer
ltsRightPercent   Summer 5.1 %
**ltsXValue      4321
**Applies only to Series with X values. See TChartSeries.AddXY method.
```

See Also

TChartLegend.LegendStyle

TChartSeries.ValueFormat

TChartSeries.PercentFormat

TChart.OnGetLegendText

TickInnerLength Property

See also

Applies to

TChartAxis component

Declaration

```
property TickInnerLength : Integer;
```

Description

Default Value: 0

The TickInnerLength property defines the length in pixels of Axis ticks drawn inside Chart boundaries.

Inner ticks are hidden by default. Set it to a value greater than zero to show the inner Axis Ticks.

Use the TickInner property to determine the Pen used to draw them.

You can combine TChartAxis.Ticks, TChartAxis.MinorTicks and TChartAxis.TicksInner at same time.

See Also

TChartAxis.TicksInner

TChartAxis.Ticks

TChartAxis.MinorTicks

TickLength Property

See also

Applies to

TChartAxis component component

Declaration

```
property TickLength : Integer;
```

Description

Default Value: 4

The TickLength property defines the length of Axis Ticks in logical pixels.

Use the Ticks Pen property to change pen attributes.

Set it to zero to hide Axis Ticks.

You can combine TChartAxis.Ticks, TChartAxis.MinorTicks and TChartAxis.TicksInner at same time.

See Also

TChartAxis.TicksInner

TChartAxis.Ticks

TChartAxis.MinorTicks

TChartAxis.TickOnLabelsOnly

TickOnLabelsOnly Property

See also

Applies to

TChartAxis component component

Declaration

```
property TickOnLabelsOnly : Boolean;
```

Description

Default True

This property sets the Axis Ticks and Axis Grid to be drawn only to coincide at Labels. Otherwise they will be drawn at all axis increment positions. When the Axis LabelsSeparation property is greater than 0 (default 10), Axis increases the increment property to avoid Axis Label overlap.

See Also

TChartAxis.TicksInner

TChartAxis.Ticks

TChartAxis.MinorTicks

TChartAxis.LabelsSeparation

Ticks Property

See also

Applies to

TChartAxis component

Declaration

```
property Ticks : TChartPen;
```

Description

The Ticks property determines the kind of Pen used to draw Axis marks along the Axis line.

Ticks position is calculated based on Axis.Increment, Axis.LabelsSeparation and Axis.LabelStyle properties.

There are three kind of ticks available: Ticks, MinorTicks and TicksInner.

You can show or hide any of them of have all of them Visible.

The TickLength property defines the length of Axis Ticks in logical pixels.

See Also

TChartAxis.Increment

TChartAxis.MinorTicks

TChartAxis.TickLength

TChartAxis.TicksInner

TicksInner Property

See also

Applies to

TChartAxis component

Declaration

```
property TicksInner : TChartPen;
```

Description

The TicksInner property determines the kind of Pen used to draw Axis marks along the Axis line. This is the same Ticks property does but lines are drawn inside Chart boundaries instead.

TicksInner position is calculated based on Axis.Increment, Axis.LabelsSeparation and Axis.LabelStyle properties.

There are three kind of ticks available: Ticks, MinorTicks and TicksInner.

You can show or hide any of them of have all of them Visible.

The TickInnerLength property defines the length of Axis TicksInner in logical pixels.

See Also

TChartAxis.Increment

TChartAxis.MinorTicks

TChartAxis.TickLength

TChartAxis.Ticks

Title Example

```
LineSeries1.Title:='Total Population';
```

Title Property (TChart)

See also

Applies to

TChart component

Declaration

```
property Title : TChartTitle;
```

Description

The Title property defines the Text and formatting properties to be drawn at Top Chart side. Use the Text property to enter the desired Title lines, set Visible to True and change the Font, Frame and Brush properties. Use the Alignment property to control text output position.

Title Property (TChartAxis)

See also

Applies to

TChartAxis component

Declaration

```
property Title : TChartAxisTitle;
```

Description

The Title property is a TChartAxis sub-component. It contains all properties for Axis Titles.

Axis Titles are a string of text drawed near Axis.

Use the Caption property to specify Axis Title text. Set the Font and Angle properties to desired format.

Title Property

Example

Applies to

TChartSeries component

Declaration

```
property Title : String;
```

Description

By default, Title is empty.

Every TChartSeries has a Title property of type String. The Title property is used in TChart.Legend to draw the series descriptions. If Title is empty, then the Series component Name will be used to draw the legend. Setting Title both at design time and runtime will force the Chart to repaint.

See Also

TChartAxis.Labels

TChart.Title

TChart.Foot

TChart.Legend

See Also

TChart.Foot

TChartTitle.Text

TitleRect Property

See also

Applies to

TChartTitle component

Declaration

```
property TitleRect : TRect;
```

Description

The TitleRect property returns the bounding rectangle coordinates for Chart.Title and Chart.Foot subcomponents. Title dimensions depend on TChartTitle.Frame and AdjustFrame properties.

You can use TitleRect to custom draw on Chart Title or Foot, or to custom handle mouse clicks over Title or Foot boundaries.

See Also

TChartTitle.Frame

TChartTitle.AdjustFrame

TitleSize Example

```
{50 pixels separation}
Chart1.TopAxis.TitleSize := 50 ;
```

TitleSize Property

Example

Applies to

TChartAxis component

Declaration

```
property TitleSize : Integer;
```

Description

The TitleSize property is 0 by default.

Therefore the space between the Axis Title and the Chart will be automatically calculated based on the Axis Title Width and Height. You can set this property both at design or runtime.

TopAxis Property

See also

Applies to

TChart component

Declaration

```
property TopAxis : TChartAxis;
```

Description

The TopAxis property determines the Labels and formatting attributes of Top Chart side. It also controls where Series points will be placed.

Every TChart component has five TChartAxis: Left, Top, Right, Bottom and Depth (Z).

The Top is pre-defined to be:

```
Horizontal := True ;
OtherSide := True ;
```

Refer to TChartAxis help topic for a complete description.

See Also

TChart

TChartAxis

TChart.BottomAxis

TChart.DepthAxis

TChart.LeftAxis

TChart.RightAxis

TopPos Property

See also

Applies to

TChartLegend component

Declaration

```
property TopPos : Integer;
```

Description

Default Value: 10

The TopPos property specifies the Legend's top position in percent of total chart height.

It's used when TChartLegend.Alignment is laLeft or laRight only. For laTop or laBottom Legend alignments, you can use the Chart's MarginTop and MarginBottom properties.

The TChart.OnGetLegendRect event can be used to supply specific Legend position and dimensions.

See Also

TChart.OnGetLegendRect

TChartLegend.HorizMargin

TChartLegend.VertMargin

TotalABS Property

Applies to
TChartValueList component

Declaration
```
property TotalABS : Double;
```

Description
Run-time and read only.

The TotalABS property returns the sum of all values in the list. The values are first converted to their absolute value.

TPieSeries, for example, uses this property to calculate the percent each Pie slice represents.

See TChartValueList.MaxValue property for more information.

Clicked Method (TSeriesMarks)

Applies to
TSeriesMarks component

Declaration
```
property Clicked(X,Y:Integer):Integer ;
```

Description
Returns the Marks index of the clicked Mark.

Bounds Method (TSeriesMarkPosition)

Applies to
TSeriesMarkPosition component

Declaration
```
property Bounds:TRect ;
```

Description
Read-only

Returns the bounding rectangle of the indexed Mark..

Example
```
With Series1.Marks.Positions do
Begin
 ShowMessage(IntToStr(Position4.Bounds.Left));
end;
```

Custom Property (TSeriesMarkPosition)

Applies to
TSeriesMarkPosition component

Declaration
```
property Custom:Boolean ;
```

Description
Enables/disables custom positioning of the indexed Mark.

Example
```
With Series1.Marks.Positions.Position4 do
Begin
  Custom:=True;
  LeftTop.x:=45;
  LeftTop.y:=45;
end;
```

Width Property (TSeriesMarkPosition)

Applies to
TSeriesMarkPosition component

Declaration
```
property Width:Integer ;
```

Description
Width of the indexed Mark.

Height Property (TSeriesMarkPosition)

Applies to
TSeriesMarkPosition component

Declaration
```
property Height:Integer ;
```

Description
Height of the indexed Mark.

LeftTop Property (TSeriesMarkPosition)

Applies to
TSeriesMarkPosition component

Declaration
```
var LeftTop:TPoint ;
```

Description
Position of the Left,Top point of the indexed Mark.

Example
```
Series1.Marks.Positions.Position4.Custom:=True;
 With Series1.Marks.Positions.Position4.LeftTop do
 Begin
   x:=45;
   y:=37;
 end;
```

ArrowFrom Property (TSeriesMarkPosition)

Applies to
TSeriesMarkPosition component

Declaration
```
var ArrowFrom:TPoint ;
```

Description
Position of the start point of the Mark connecting Arrow.

Example
```
Series1.Marks.Positions.Position4.Custom:=True;
With Series1.Marks.Positions.Position4.ArrowFrom do
Begin
  x:=45;
  y:=37;
end;
```

ArrowTo Property (TSeriesMarkPosition)

Applies to
TSeriesMarkPosition component

Declaration
```
var ArrowTo:TPoint ;
```

Description
Position of the end point of the Mark connecting Arrow.

Example
```
Series1.Marks.Positions.Position4.Custom:=True;
With Series1.Marks.Positions.Position4.ArrowTo do
Begin
  x:=45;
  y:=37;
end;
```

Positions Property (TSeriesMarks)

Applies to
TSeriesMarks component

Declaration
```
property Positions: TSeriesMarksPositions;
```

Description
Use this property to access Custom position characteristics for Series Marks.

Automatic method (TSeriesMarksPositions)

Applies to
TSeriesMarks component

Declaration
```
procedure Automatic( Index : Integer);
```

Description
Enable automatic positioning for the indexth Mark.

Example
```
Series1.Marks.Positions.Automatic(3);
Chart1.Repaint;
```

Position Property (TSeriesMarksPositions)

Applies to
TSeriesMarks component

Declaration
```
property Position Index : Integer : TSeriesMarksPosition;
```

Description
Set the position for the indexth Mark.

Transparent Property (TChartWall)

Applies to
TChartWall component

Declaration
```
property Transparent : Boolean;
```

Description
Default Value: False

The Transparent property controls if Walls will be filled or not.

Transparent Property (TSeriesMarks)

See also

Applies to

TSeriesMarks component

Declaration

```
property Transparent : Boolean;
```

Description

Default Value: False

The Transparent property controls if Series Marks background will be filled or not.

See Also
TSeriesMarks.BackColor

UnClipRectangle Method

See also

Applies to
TChart component

Declaration
```
Procedure UnClipRectangle;
```

Description
(Advanced)

The UnClipRectangle method removes any clipping region applied to Chart Canvas. Clipping regions are used to prevent drawing to escape from desired rectangle coordinates. It is used internally to keep Marks and Series points to display outside ChartRect rectangle.

Note: Metafiles do not accept clipping regions. Metafile format is used when printing, so clipping can not be performed.

See Also
TChart.ClipRectangle

UndoZoom Method

See also

Applies to

TChart, TDBChart components

Declaration

```
procedure UndoZoom ;
```

Description

The Chart.UndoZoom method will rescale the Chart Axis to their Maximum and Minimum values.Each Axis "depends" on one or more Series components, so Axis values will depend on their associated Series Maximum and Minimum values. UndoZoom also restores any runtime Scrolling.

See Also

Chart Zoom

Chart Scrolling and Panning

UsePatterns Property

See also

Applies to

TPieSeries component

Declaration

```
property UsePatterns : Boolean
```

Description

Default False

The UsePatterns property indicates, when True, that Pie Sectors will be filled using different Brush pattern styles. There are 6 different pattern styles.

See Also

TChart.SetBrushCanvas

TChartSeries.ValueColor

PatternPalette global constant

GetDefaultPattern global function

UseYOrigin Property

See also

Applies to

TCustomBarSeries component

Declaration

```
property UseYOrigin : Boolean;
```

Description

Default Value: True

The UseYOrigin property defines if Bars will be bottom aligned to the YOrigin property value.

When False, the minimum of all Bar values is used as the Bar origins value.

When True, the YOrigin property is used as the start point for Bars.

See Also
TCustomBarSeries.YOrigin

Value Property

See also Example

Applies to

TChartValueList component

Declaration

```
property Value[ Index:Longint ] : Double;
```

Description

Run-time only.

The Value array property holds all values in the list. The Series component repaints whenever values are changed. You can use the Value property to retrieve or change values:

Value property Example

This code changes the 6th series point value to 25:

```
LineSeries1.YValues.Value[ 5 ] := 25 ;
LineSeries1.Repaint;
```

This code is equivalent:

```
LineSeries1.YValues[ 5 ] := 25 ;
LineSeries1.Repaint;
```

This code too (repaints automatically):

```
LineSeries1.YValue[ 5 ] := 25 ;
```

See Also

TChartSeries.AddY

TChartSeries.Delete

ValueColor Example

You can change by coding the point's color. The following code will change the (25+1)th point color to clGreen.

```
LineSeries1.ValueColor[ 25 ] := clGreen ;
```

Or...

The following code will change all negative values to be red and the positive ones to be blue.

```
LineSeries1.ColorEachPoint := True;
for t := 0 to LineSeries1.Count - 1 do
  if LineSeries1.YValue[ t ] < 0 then
    LineSeries1.ValueColor[ t ] := clRed
  else
  LineSeries1.ValueColor[ t ] := clBlue;
```

That can be also done in most Series types.

ValueColor Property

Example

Applies to

TChartSeries component

Declaration

```
property ValueColor[Index:Longint]:TColor
```

Description

The TChartSeries ValueColor property is an array of TColor values. Each color value corresponds to a different point in the Series.

ValueFormat Example

```
Chart1.LeftAxis.AxisValuesFormat := '###0.0###';
```

See Delphi help under FormatFloat Function for complete details.

ValueFormat Property

Example

Applies to

TChartAxis, TChartSeries components

Declaration

```
property ValueFormat : String;
```

Description

ValueFormat is a standard Delphi formatting string specifier.

Chart Axis uses it to draw the axis labels.

Chart Series uses it to draw the Marks.

ValueList Example

TBubbleSeries component has a 3rd ValueList: RadiusValues

This "extra" ValueList is the same as:

```
BubbleSeries.ValueList[ 2 ]
```

ValueList Property

Example

Applies to

TChartSeries component

Declaration

```
property ValueList[Index:Longint] : TChartValueList
```

Description

All Series have at least 2 (two) ValueList's:

- XValues (the 0 index ValueList)
- YValues (the 1 index ValueList)

You dont need normally to use ValueList directly. Series that have more than 2 ValueList's normally "publish" the new ValueLists for you to access them as properties.

ValueMarkText Property

Applies to
TChartSeries component

Declaration
```
property ValueMarkText[Index:Longint]:String
```

Desciption
Returns the String representation of a Index point used to draw the Mark.

ValueSource Example

Being Quantity, a former numeric Field Name in Table1.

```
LineSeries1.DataSource := Table1 ;
LineSeries1.YValues.ValueSource := 'Quantity' ;
```

Case 2. A Chart Series is connected to another Chart Series:

If the Series which owns this ValueList is attached to a TChart (or a TDBChart), and if the Series.DataSource property is any TChartSeries component, then the ValueSource property must refer to an existing TChartValueList Name in the underlying TChartSeries.

Example:

```
LineSeries1.DataSource:=BubbleSerie2;
LineSeries1.YValues.ValueSource:='Y';
```

Being Y, the BubbleSeries2.YValues.Name.

ASSUMPTIONS:

Changing the ValueSource property both at design or runtime, will force the Series to fill again its values from the specified DataSource.

This must be specially considered when the DataSource Series property is a database TTable or TQuery or TClientDataset, because maybe it takes a long time to retrieve all records.

If the dataset is closed, the automatic record retrieving will be performed later on, when the dataset is opened again.

ValueSource Property

Example

Applies to

TChartValueList component

Declaration

```
property ValueSource : String;
```

Description

Each Chart Value List has a ValueSource property. You can set this string property to different kind of values:

Case 1. A Chart Series is connected to a database source:

If the Series that owns this ValueList is attached to a TDBChart, and the Series.DataSource property is a TTable, TQuery, TClientDataset or any TDataSet component which provides database records, then the ValueSource property must refer to an existing numeric, date, time or datetime Field Name in the underlying dataset.

Example:

```
LineSeries1.DataSource := Table1 ;
LineSeries1.YValues.ValueSource := 'Quantity' ;
```

Being Quantity, a former numeric Field Name in Table1.

Case 2. A Chart Series is connected to another Chart Series:

If the Series which owns this ValueList is attached to a TChart (or a TDBChart), and if the Series.DataSource property is any TChartSeries component, then the ValueSource property must refer to an existing TChartValueList Name in the underlying TChartSeries.

Example:

```
LineSeries1.DataSource:=BubbleSerie2;
LineSeries1.YValues.ValueSource:='Y';
```

Being Y, the BubbleSeries2.YValues.Name.

ASSUMPTIONS:

Changing the ValueSource property both at design or runtime, will force the Series to fill again its values from the specified DataSource.

This must be specially considered when the DataSource Series property is a database TTable or TQuery or TClientDataset, because maybe it takes a long time to retrieve all records.

If the dataset is closed, the automatic record retrieving will be performed later on, when the dataset is opened again.

ValuesListCount Method

Applies to
TChartSeries component

Declaration
```
function ValuesListCount:Longint;
```

Description
This Function returns the number of ValueLists for the Series. Normally, ValuesListCount is 2 (XValues and YValues) but there are some Series that have more ValuesList:

BubbleSeries has 3 ValuesLists (X, Y and Radius).

GanttSeries has 3 ValuesList (Y, Start and End).

Variables list

See also

Description

This table lists the different input variables permitted by each Series type. Some variables, such as Label fields, are optional.

SERIES TYPE	DATASOURCE PROPERTIES
Basic	
Line	XValues, YValues, XLabel
Fast Line	XValues, YValues, XLabel
Bar	XValues, YValues (called Bar), XLabel
Area	XValues, YValues, XLabel
Point	Xvalues, YValues, XLabel
Pie	PieValues, XLabel
Arrow	StartXValues, StartYValues, XLabel, EndXValues, EndYValues
Bubble	Xvalues, YValues, XLabel, RadiusValues
Gantt	StartValues, EndValues, AY (Y axis level), AXLabel (Label optionally shown on Y-axis or as mark)
Shape	X0 (Top), Y0 (Bottom), X1 (Left), Y1 (Right)

See Also
Series unit

VertAxis Example

```
BubbleSeries1.VertAxis := aRightAxis ;
```

VertAxis Property

See also Example

Applies to

TChartSeries component

Declaration

```
property VertAxis : TVertAxis
```

Description

The VertAxis property is of type: TVertAxis. It means by which Vertical Axis (Left or Right axis) will be the Series vertically scaled. You can change the desired Vertical Axis both at design and runtime:

See Also
TChartSeries.HorizAxis

VertMargin Property

See also

Applies to

TChartLegend component

Declaration

```
property VertMargin : Integer
```

Description

Default 0

The VertMargin property determines the amount of pixels that Legend will reduce Chart rectangle. The Legend.ResizeChart property must be True and Legend.Alignment property must be laTop or laBottom. When 0, the corresponding Chart margin property is used to determine the amount of pixels for margins (Chart.MarginTop for laTop Legend.alignment and Chart.MarginBottom for laBottom Legend.Alignment).

See Also

TChart.MarginBottom

TChart.MarginTop

TChartLegend.Alignment

TChartLegend.HorizMargin

TChartLegend.ResizeChart

VertSize Property

See also

Applies to

TSeriesPointer component

Declaration

```
property VertSize : Integer;
```

Description

Default Value: 4

The VertSize property specifies the Series Pointer height in logical pixels.

Series that derive from TPointSeries usually override the HorizSize and VertSize properties.

For example, TBubbleSeries uses the Radius property to determine the correct HorizSize and VertSize, so these properties have no effect in that Series.

See Also

TSeriesPointer

TSeriesPointer.HorizSize

View3D Property

Applies to
TChart, TDBChart components

Declaration
```
property View3d : Boolean
```

Description
Default value: True

View3D will draw each Series simulating a 3D effect.You can control the 3D proportion by using Chart.Chart3DPercent property. Chart.View3DWalls depends on View3D property.

View3DWalls Property

Applies to
TChart, TDBChart components

Declaration
```
property View3dWalls : Boolean
```

Description
Default value: True

View3DWalls will draw Left and Bottom "walls" to simulate 3D effect. You can control the 3D Wall proportion by using Chart.Chart3DPercent property. Chart.View3D controls (on/off) View3DWalls property.

View3DOptions Property

Applies to
TChart, TDBChart components

Declaration
```
property View3dOptions : TView3dOptions
```

Description
View3DOptions control access for Rotation, Zoom and scrolling.

Visible Property

See also Example

Applies to

TChartAxis component

Declaration

```
property Visible : Boolean
```

Description

This boolean property Shows / Hides the Axis lines, ticks, grids, labels and title. You can change it both at design and runtime:

Visible Property (TChartGradient)

See also Example

Applies to
TChartGradient component

Declaration
```
property Visible : Boolean;
```

Description
Default False

The Visible determines whether the gradient fill appears on screen.

Visible Property (TChartLegend)

Applies to
TChartLegend component

Declaration
```
property Visible : Boolean;
```

Description
Default Value: True

The Visible property determines if Chart Legend will be displayed or not.

Visible Property (TChartPen)

Applies to
TChartPen component

Declaration
```
property Visible : Boolean;
```

Description
The Visible property determines if the pen will draw lines or not.

For example, TChart.Frame can be hidden by setting it's Visible property to False:

```
Chart1.Frame.Visible:=False;
```

Visible Property (TChartTitle)

Applies to
TChartTitle component

Declaration
```
property Visible : Boolean;
```

Description
The Visible property controls if Chart Title or Foot will be displayed or not. When False, Chart dimensions grow to fit all available space.

Visible Property (TSeriesMarks)

See also

Applies to

TSeriesMarks component

Declaration

```
property Visible : Boolean;
```

Description

The Visible property controls if Series Marks will be displayed or not.

Visible Property

See also

Applies to
TSeriesPointer component

Declaration
```
property Visible : Boolean;
```

Description
The Visible property controls if Series pointers will be displayed or not.

When using a TPointSeries (or any Series component derived from TPointSeries), setting Visible to False will not display anything.

Pointers can be useful with TLineSeries or TAreaSeries.

When points are Visible, an extra margins are applied to the four chart axis (Left, Right, Top and Bottom).

This is to make points just at axis limits to be shown.

You can deactivate this extra margins by setting the TSeriesPointer.InflateMargins property to False.

Points are filled using TSeriesPointer.Brush property.

Visible Example

```
Chart1.BottomAxis.Visible := False ;
```

See Also
Chart.AxisVisible

Visible property (TChartGradient) Example

This code creates a gradient fill:

```
Chart1.Gradient.Visible := True ;
Chart1.Gradient.StartColor := clYellow ;
Chart1.Gradient.EndColor := clBlue ;
Chart1.Gradient.Direction := gdLeftRight ;
```

See Also

TChartGradient.Direction

TChartGradient.EndColor

TChartGradient.StartColor

See Also

TSeriesMarks.Transparent

TSeriesMarks.BackColor

See Also

TSeriesPointer.InflateMargins

TSeriesPointer.Brush

VisibleCount Method

Applies to
TChartSeries component

Declaration
```
function VisibleCount : Longint;
```

Description
This function returns the number of Visible points in the Series. These are the points which X coordinate is between the Minimum and Maximum horizontal Axis range. To iterate, you need to know the FirstValueIndex and LastValueIndex points.

WarningHighColor Method

See also

Applies to

TChart component

Declaration

```
procedure WarningHighColor;
```

Description

This method is used internally at design-time to inform the developer that some property values will look much better on Screen if the video card mode is set to 16k colors or True Color 24bit / 32bit. It calls the Chart.IsScreenHighColor function and, when False, pops up a message dialog notifying the developer. You can use this method at run-time in your own applications if you need the same video checking.

See Also
TChart.IsScreenHighColor

Welcome to TeeChart !

The 100% Native Data-Aware Charting Component Library for Delphi and C++ Builder.

Helpfile revision 35th May 1999.

Width Property (TChartPen)

Applies to
TChartPen component

Declaration
```
property Width : Integer;
```

Description
The Width property determines the width of lines the pen draws.

Warning:
When drawing to Screen, the Width is expressed in pixels. When drawing to Printer or to Metafile canvases, the Pen Width will be converted based on TChart.PrintResolution property.

Windows GDI has a special Width value of "0", that corresponds to the minimum pen width the drawing device supports.

Width3D Example

This code draws an horizontal "frame":

```
With Chart1,Canvas do
begin
  MoveTo( ChartRect.Left , ChartRect.Bottom - 10 ) ;
  LineTo( ChartRect.Right , ChartRect.Bottom - 10 ) ;
  LineTo( ChartRect.Right + Width3D , ChartRect.Bottom - 10 - Height3D ) ;
end;
```

Width3D Property

See also Example

Applies to

TChart component

Declaration

```
property Width3D : Longint;
```

Description

Run-time and read only. The Width3D property determines the width in pixels of the Chart 3D effect. The Chart.View3D property should be True. It equals zero if Chart.View3D is False. It changes when changing the Chart.Chart3DPercent property.

See Also

TChart.Height3D

TChart.SeriesHeight3D

TChart.SeriesWidth3D

XLabel Property

Applies to
TChartSeries component

Declaration
```
property XLabel[Index:Longint] : String;
```

Description
This array property contains the horizontal Series Labels. You can access the individual Label strings by using the Index parameter. Index must be between 0 and Count - 1 If you modify an XLabel, the Series will be repainted to reflect any changes.

XLabelText Method

Applies to
TChart, TDBChart components

Declaration
```
function XLabelText ( ASeries : TChartSeries ; ValueIndex : Longint ) :
String ;
```

Description
This function returns the Series.XLabel text of the ValueIndexth Series point.

XLabels Example

If a LineSeries is made of pairs of points, each with X and Y values, you can choose to have or to not have a corresponding horizontal label for each point X values.

This will add a new point to LineSeries1 with 3.5 (X) and 5.2 (Y) values, no associated horizontal label and red color.

This point will be inserted in its horizontal position.

```
LineSeries1.AddXY( 3.5 , 5.2, '', clRed);
```

You can also have the opposite case:

This will append a new point to LineSeries1, with an YValue of 5.2 and a 'Manhattan' horizontal label in yellow color.

```
LineSeries1.AddY( 5.2 , 'Manhattan', clYellow);
```

Last case is applicable to Bar Series and Pie Series.

XLabels Property

See also Example

Applies to

TChartSeries component

Declaration

```
property XLabels : TList;
```

Description

The XLabels property is a string list used to store the corresponding horizontal labels for a specific chart Series at runtime. Its use is optional. You can have both XValues and XLabels or a single one only. How axis labels and Series Marks will be displayed depends on its own respective style properties.

See Also
Series XLabelsSource property

XLabelsSource Property

See also

Applies to

TChartSeries component

Declaration

```
property XLabelsSource : String;
```

Description

The XLabelsSource is the name of a DataSet field. DataSet can be a TTable, TQuery, TClientDataset or any other TDataSet derived component. This is the database field that contains the point Labels. Valid Field types are all that have a valid AsString method. This property is optional. If no LabelsSource specified, Series XLabels will be empty.

See Also
Series XLabels

XRadius Property

See also

Applies to

TCircledSeries component

Declaration

```
property XRadius : Longint
```

Description

Run-time and read only. The XRadius property returns the exact ellipse's radius horizontal size in pixels. The ellipse XRadius and YRadius can be set to a fixed number of pixels by using the CircledSeries.CustomXRadius and CustomYRadius. The CircledSeries.Circled property controls if both radius must be proportional to the Screen X/Y ratio.

See Also

TCircledSeries.YRadius

TCircledSeries.CustomXRadius

TCircledSeries.CustomYRadius

XScreenToValue Example

You can, for example, use this function to know the Value where the user mouse-clicked:

```
Procedure TForm1.Chart1OnClick( ... x,y ... ) ;
Begin
  ShowMessage('You clicked at:
  '+FloatToStr( LineSeries1.XScreenToValue(x) ));
end ;
```

XScreenToValue Method

See also Example

Applies to

TChartSeries component

Declaration

```
function XScreenToValue(ScreenPos:Longint):Double;
```

Description

This function returns the numeric Value that corresponds to the specified Horizontal Screen coordinate. The resulting Value is based on the Series.GetHorizAxis

See Also
YScreenToValue

XValue Property

Applies to
TChartSeries component

Declaration
```
property XValue[Index:Longint] : Double;
```

Description
XValue array property returns the Index value in the XValues List

XValueToText Method

See also

Applies to

TChartSeries component

Declaration

```
function XValueToText( Const AValue : Double ) : String;
```

Description

The XValueToText function returns the formatted text representation of the specified AValue parameter.

The AValue parameter is considered to be an horizontal X coordinate.

It returns the text string corresponding to an hypothetic horizontal Axis Label for that value.

It calls the corresponding horizontal axis LabelValue function.

See Also

TChartAxis.LabelValue

TChartSeries.YValueToText

XValues Example

Many things can be done with the XValues property.

For example, you can modify its values (thus forcing the Chart to repaint):

This will change the 35th point horizontal value to 10.

```
LineSeries1.XValues.Value[ 35 ] := 10 ;
```

You do not need to use 'Value' :

```
eg.
For index := 0 to Series1.Count - 1 do
 Series1.XValues[Index] := Series1.XValues[Index] /10;
```

Or, if contains DateTime XValues...

```
LineSeries1.YValues.Value[ 35 ] := EncodeDate( 1995 , 4 , 13 );
```

Or...

```
With LineSeries1.XValues do Value[ 35 ] := Value[ 35 ] + 1 ;
```

XValues Property

See also Example

Applies to

TChartSeries component

Declaration

```
property XValues:TChartValueList
```

Description

By default, any TChartSeries has an XValues property. This is the TChartValueList where the point values will be stored at runtime. Also by default, XValues is a Public property. Some derived Series publish it: TLineSeries, TBarSeries, TPointSeries, etc. Some others publish it with another, more friendly name: TGanttSeries.StartValues, etc.

WARNING:

You CAN NOT Delete, Clear or Add values DIRECTLY. You need to call the TChartSeries equivalent methods to do this.

See Also

TChartValueList

YValues

Y Percents Explanation

For each Series point, its corresponding value is calculated with the following formula:
`percent:=100.0*value/TotalABS;`
Being TotalABS the sum of all point's values.

Example:
Having the following point values (either vertical, horizontal):

5, 8, -3, 6, -1, 0

The first point percentual value is:

100.0 * 5 / 23 --> 21.739 %

Where 23 equals: 5 + 8 + abs(-3) + 6 + abs(-1) + 0

YOrigin Property

See also

Applies to

TCustomBarSeries component

Declaration

```
property YOrigin : Double;
```

Description

Default Value: 0

The YOrigin property determines the axis value used as a common bottom for all Bar points.

The UseYOrigin property must be True (the default) to use the YOrigin property.

Bars with a value bigger than YOrigin are drawn in one direction and Bars with a lower value are drawn in the opposite direction.

This applies both to TBarSeries and THorizBarSeries components.

See Also
TCustomBarSeries.UseYOrigin

YRadius Property

See also

Applies to

TCircledSeries component

Declaration

```
property YRadius : Longint
```

Description

Run-time and read only. The YRadius property returns the exact ellipse's radius vertical size in pixels. The ellipse XRadius and YRadius can be set to a fixed number of pixels by using the CircledSeries.CustomXRadius and CustomYRadius. The CircledSeries.Circled property controls if both radius must be proportional to the Screen X/Y ratio.

See Also

TCircledSeries.XRadius

TCircledSeries.CustomXRadius

TCircledSeries.CustomYRadius

YScreenToValue Example

You can, for example, use this function to set the Axis Minimum Value to a specific current Y Screen coordinate.

```
LineSeries1.GetVertAxis.Minimum := LineSeries1.YScreenToValue ( 123 );
```

YScreenToValue Method

See also Example

Applies to

TChartSeries component

Declaration

```
function YScreenToValue(ScreenPos:Longint) : Double;
```

Description

This function returns the numeric Value that corresponds to the specified Vertical Screen coordinate. The resulting Value is based on the Series.GetVertAxis

See Also
XScreenToValue

YValue Property

Applies to
TChartSeries component

Declaration
```
property YValue[Index:Longint]:Double
```

Description
YValue array property returns the Index value in the YValues List

YValueToText Method

See also

Applies to

TChartSeries component

Declaration

```
function YValueToText( Const AValue : Double ) : String;
```

Description

The YValueToText function returns the formatted text representation of the specified AValue parameter.

The AValue parameter is considered to be a vertical Y coordinate.

It returns the text string corresponding to an hypothetic vertical Axis Label for that value.

It calls the corresponding vertical axis LabelValue function.

See Also

TChartSeries.XValueToText

TChartAxis.LabelValue

YValues Property (TChartSeries)

See also Example

Applies to

TChartSeries component

Declaration

```
property YValues : TChartValueList;
```

Description

By default, any TChartSeries has an YValues property. This is the TChartValueList where the point values will be stored at runtime. Also by default, YValues is a Public property.

Some derived Series publish it: TLineSeries, TBarSeries, TPointSeries, and some others publish it with another, more friendly name: TPieSeries.PieValues, etc.

WARNING:

You CAN'T Delete, Clear or Add values DIRECTLY. You need to call the TChartSeries equivalent methods to do it.

YValues property (TChartSeries) Example

Many things can be done with the YValues property.

For example, you can modify its values (thus forcing the Chart to repaint):

```
LineSeries1.YValues.Value[35]:=10;
```

That will change the 35th point vertical value to 10

Or...

```
With LineSeries1.YValues
  Value[35] := Value[35]+1;
```

Or, if contains DateTime YValues...

```
LineSeries1.YValues.Value[35] := EncodeDate(1995,4,13);
```

See Also

Assigning a DataSource to YValues.

XValues

ZOrder Property

See also

Applies to

TChartSeries component

Declaration

```
property ZOrder : LongInt;
```

Description

Read-only and run time.

The ZOrder property returns at which position along the depth axis the Series is drawn.

It's valid only when TChart.View3D property is True and when there's more than one Series in same chart.

You can't alter the ZOrder property directly. If you want a different order you need to use the TChart.SeriesList property instead.

The ZOrder property is calculated for each Series just before the Chart is drawn.

The TChart.ApplyZOrder property controls if Series will be assigned a different Z position or not. When False, all Series are drawn at same Z plane.

The TChart.MaxZOrder property returns the highest of all Series ZOrder values.

See the "LastValu.PAS" example unit for a demo of ZOrder.

See Also

TChart.ApplyZOrder

TChart.SeriesUp

TChart.MaxZOrder

ZoomPercent Example

This sets AnimatedZoom to True, performs a Zoom In and a Zoom Out:

```
Chart1.AnimatedZoom:=True;
Chart1.ZoomPercent( 125 ); { Zoom IN 125% }
Chart1.ZoomPercent( 75 );  { Zoom OUT 75% }
```

ZoomPercent Method

See also Example

Applies to

TChart component

Declaration

```
procedure ZoomPercent( Const PercentZoom : Double );
```

Description

The Chart.ZoomPercent method applies the specified PercentZoom Zoom In/Out to the current Axis scales. When PercentZoom is greater than 100%, Zoom Out is performed. When PercentZoom is lower than 100%, Zoom In is performed. The AnimatedZoom property controls if Zoom is done directly in only one step or by multiple zooms thus giving an animation effect.

See Also

TChart.AllowZoom

TChart.AnimatedZoom

TChart.AnimatedZoomSteps

TChart.OnUndoZoom

TChart.OnZoom

TChart.ZoomRect

TChart.Zoomed

TChartAxis.SetMinMax

ZoomRect Method

See also

Applies to

TChart component

Declaration

```
procedure ZoomRect(Const Rect : TRect);
```

Description

The ZoomRect method does zoom on Chart contents. The Rect parameter defines the pixel rectangle coordinates of the chart area to be zoomed. Calling this method is similar to manually drawing a zoom box at run-time by dragging the left mouse button.

You can perform a zoom in by specifying a Rect parameter enclosed on Chart dimensions, and zoom out by specifying a bigger rectangle parameter.

Zoom in example:

```
Chart1.ZoomRect( Rect( 5,5,Chart1.Width-5,Chart1.Height-5 ) );
```

Zoom out example:

```
Chart1.ZoomRect( Rect( -5,-5,Chart1.Width+5,Chart1.Height+5 ) );
```

See Also

TChart.AllowZoom

TChart.AnimatedZoom

TChart.AnimatedZoomSteps

TChart.OnUndoZoom

TChart.OnZoom

TChart.UndoZoom

TChart.Zoomed

TChart.ZoomPercent

TeeZoomMouseButton global variable

AnimatedZoomFactor global variable

Zoomed Property

See also

Applies to

TChart component

Declaration

```
property Zoomed : Boolean;
```

Description

Run-time only.

The Zoomed property determines if Chart axis scales do not fit all Chart points.

It is set to True when users's apply zoom or scroll to the Chart using the mouse at run-time.

The UndoZoom method sets the Zoomed property to False and resets the axis scales to fit all Series points.

The default value is True, meaning no zoom or scroll has been applied to the chart after it has been displayed for first time.

See Also

TChart.OnScroll

TChart.OnZoom

TChart.UndoZoom

TChart.ZoomPercent

TChart.ZoomRect

TeeLineSeparator Global Constant

Applies to
Global Constant

Declaration
```
Const TeeLineSeparator = ascii character #13;
```

Description
Use this character constant for multiple line Labels when adding Points to a data Series.

eg.
```
Series1.Add(1234, 'My two' + TeeLineSeparator + 'line
    label',clRed);
```
See also the Axis LabelsMultiLine property.

clTeeColor Color Value Property

Applies to
TChart, TDBChart components

Declaration
```
Const clTeeColor = clScrollBar;
```

Description
Most Color properties in TeeChart component, accept the clTeeColor constant value. That is equal to Delphi clScrollBar (or -1, in numeric).

It is a very important feature to have clTeeColor:

When adding new points to a Chart Series, specifying clTeeColor as the point color, it will make TeeChart assigns a different color to each point. (See TChartSeries.ColorEachPoint property for more information).

When attaching new series to any Chart, having the TChartSeries.SerieColor property to clTeeColor it will make TeeChart assigns a different color to each Series.

Some Series types allow clTeeColor in their Pen and Brushes properties, thus forcing to use the actual point color instead of the Pen or Brush assigned color.

Alignment Property (TChartShape)

Applies to
TChartShape component

Declaration
```
property Alignment : TAlignment;
```

Description
The Alignment property decides the alignment of the text of a TChartShape.

Bounds property (TChartShape) Example

The following code uses TChartShape.Bounds to draw a diagonal cross over a TChartShape series:

```
{ Drop a TChart and a Shape Series. Rename Series1 to Shape1 }
procedure TForm1.Chart1AfterDraw(Sender: TObject);
begin
 With Chart1.Canvas do
 begin
  Pen.Color:=clRed;
  Pen.Width:=2;
  Pen.Style:=psSolid;
  With Shape1.Bounds do
  begin
   MoveTo( Left, Bottom );
   LineTo( Right, Top );
   MoveTo( Left, Top );
   LineTo( Right, Bottom );
  end;
 end;
end;
```

Bounds Property (TChartShape)

See also Example

Unit
TeeShape

Applies to
TChartShape component

Declaration
```
property Bounds;
```

Description
The Bounds property returns the biggest rectangle enclosing TChartShape, in logical pixels. It accesses TChartShape.GetShapeRect method.

See Also
TChartShape.GetShapeRect

Brush Property (TChartShape)

See also

Applies to

TChartShape component

Declaration

```
property Brush : TBrush;
```

Description

The Brush property defines the kind of brush used to fill shape background.

See Also
TChartShape.Pen

Font property TChartShape Example

This code sets Shape text and font:

```
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
 { Drop a TChart and a TChartShape components and rename the Shape
  to "MyShape" }
 With MyShape do
 begin
  Style:=chasDiamond ;
  Font.Size:=12;
  Font.Color:=clBlue;
  Font.Style:=[fsBold];
  Text.Clear;
  Text.Add('Hello');
  Text.Add('World');
 end;
end;
```

Font Property (TChartShape)

See also Example

Applies to

TChartTitle component

Declaration

```
property Font : TFont;
```

Description

The Font property determines the font attributes used to output TChartShape.Text strings. No auto font sizing is performed, so you must specify the desired font size to avoid shape text to go over Shape boundaries.

See Also
TChartShape.Text

GetShapeRect Method

See also

Applies to

TChartShape component

Declaration

```
function GetShapeRect : TRect;
```

Description

The GetShapeRect method calculates and returns the TChartShape bounding rectangle coordinates.

You can use TChartShape.Bounds property instead, as Bounds uses GetShapeRect as the property get method.

See Also
TChartShape.Bounds

Pen Property (TChartShape)

See also

Applies to

TChartShape component

Declaration

```
property Pen : TPen;
```

Description

The Pen property speficies the pen used to draw the shape.

See Also
TChartShape.Brush

RoundRectangle Property

See also

Applies to
TChartShape component

Declaration
```
property RoundRectangle : Boolean;
```

Description
Default : False

The RoundRectangle property determines if TChartShape draws rounded rectangle corners. It has effect only when shape Style is chasRectangle.

See Also
TChartShape.Style

Style Property (TChartShape)

Example

Applies to

TChartShape component

Declaration

```
property Style : TChartShapeStyle
```

Description

Default Value: chasHorizLine

The Style property defines how a TChartShape component appears on a Chart.

These are the possible values and their meanings:

chasRectangle	The shape is a rectangle
chasCircle	The shape is a circle
chasVertLine	The shape is a vertical line
chasHorizLine	The shape is an horizontal line
chasTriangle	The shape is a triangle
chasInvertTriangle	The shape is an inverted triangle
chasLine	The shape is a line
chasDiamond	The shape is a diamond

Style property (TChartShape) Example

You can change the Shape style both at design and run-time.

```
MyShape.Style := chasTriangle ;
```

TChartShape Component

Properties Methods

Unit

TeeShape

Ancestor

TChartSeries

Description

The TChartShape component is a special Series component. It allows the developer to place shapes inside Chart bounds. It works very similar to Delphi standard TShape component. To see a visual representation of this Series type, go to the TeeChart User Guide.

Set the ParentChart property to the desired Chart component.

Then, set the X0, Y0 and X1, Y1 properties to the desired point coordinates. The X0, Y0, X1 and Y1 properties must be expressed in axis scales.

The default Shape style is an horizontal line. Choose the desired style (Rectangle, Ellipse, etc) by using the Style property.

The Brush and Pen properties determine the shape color and frame attributes.

The Clicked method can be used to check if mouse cursor is over shape boundaries.

Shapes can be zoomed and scrolled as with any other Series type.

Please refer to TChartSeries ancestor description for all common Series properties like Marks, Axis dependence, methods and events.

Methods

Key Methods

GetShapeRect

Properties

Run-time only

Key Properties

Alignment

Brush

Font

Pen

RoundRectangle

Style

Text

Transparent

X0

X1

XYStyle

Y0

Y1

TChartShapeStyle Type

Unit
TeeShape

Declaration
```
TChartShapeStyle = (chasRectangle, chasCircle, chasVertLine, chasHorizLine,
chasTriangle, chasInvertTriangle, chasLine, chasDiamond);
```

Description
TChartShapeStyle defines the list of possible values for the TChartShape.Style property.

chasRectangle	The shape is a rectangle
chasCircle	The shape is a circle
chasVertLine	The shape is a vertical line
chasHorizLine	The shape is an horizontal line
chasTriangle	The shape is a triangle
chasInvertTriangle	The shape is an inverted triangle
chasLine	The shape is a line
chasDiamond	The shape is a diamond

TeCanvas Unit

The TeCanvas unit contains the declarations for the following components and for the enumerated type associated with them. When you add a component declared in this unit to a form, the unit is automatically added to the uses clause of that form's unit. The following items are declared in the this unit:

Components
TCanvas3D

TeeShape Unit

The TeeShape unit contains the declarations for the following components and for the enumerated type associated with them. When you add a component declared in this unit to a form, the unit is automatically added to the uses clause of that form's unit. The following items are declared in the this unit:

Components
TChartShape

Types
TChartShapeStyle

Routines
To see a listing of items declared in this unit including their declarations, use the Project Brower.

Text property (TChartShape) Example

```
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
 { Drop a TChart and a TChartShape components and rename the Shape
  to "MyShape" }
 With MyShape do
 begin
  Style:=chasDiamond ;
  Font.Size:=12;
  Font.Color:=clBlue;
  Font.Style:=[fsBold];
  Text.Clear;
  Text.Add('Hello');
  Text.Add('World');
 end;
end;
```

Text Property (TChartShape)

See also Example

Applies to

TChartShape component

Declaration

```
property Text : TStrings;
```

Description

The Text property is used by TChartShape component to display customized strings inside Shapes.

You can use the Font and Aligment properties to control Text display.

Note: You would maybe need to change Shape Font size to a different value when creating metafiles or when zooming Charts.

See Also

TChartShape.Font

TChartShape.Alignment

Transparent Property (TChartShape)

Applies to
TChartShape component

Declaration
```
property Transparent : Boolean;
```

Description
Default : False

The Transparent property controls if TChartShape components will use the Shape Brush attributes to fill the interior of the Shape.

When False, Shapes do not redraw their background so charting contents behind Shape Series is seen inside the Shape.

X0, Y0, X1, Y1 Properties

See also

Applies to

TChartShape component

Declaration

```
property X0 : Double;
property Y0 : Double;
property X1 : Double;
property Y1 : Double;
```

Description

These properties define the Top - Left and Bottom - Right coordinates of the englobing TChartShape rectangle.

The values should be expressed in Axis coordinates.

You can convert from Screen pixel coordinates to values and vice-versa using several TChart and TChartSeries methods like XScreenToValue and YScreenToValue.

See Also

TChartShape.Style

TChartSeries.XScreenToValue

TChartSeries.YScreenToValue

XYStyle Property

Applies to
TChartShape component

Declaration
```
property XYStyle : TChartShapeXYStyle;
```

Description
Default : xysAxis

Can be xysAxis, xysPixels or xysAxisOrigin.

Example
```
With Series1 do
begin
 XYStyle:=xysAxisOrigin
 X0 := EncodeDate(1998,1,1); // <-- in BottomAxis scales
 Y0 := 1234.45; //<-- in LeftAxis scales
 X1 := 100; //<-- 100 pixels width
 Y1 := 30; //<-- 30 pixels height
end;
```

FunctionType property Example

To set the period for Function you should use the FunctionType property

To define a functioned series by code you should first create a new series for the function. The series may be of any type.

```
{ Set the function using the SetFunction method}.
Series1.SetFunction(TAddTeeFunction.Create(Self));
{You may then define the period for the function - here setting it to to 5}
Series1.FunctionType.Period:=5;
```

To undefine (delete) a function defined for the series use

```
Series1.SetFunction(nil);
```

Weighted Property

See also

Applies to

TAverageTeeFunction component

Declaration

```
property Weighted : Boolean;
```

Description

Default Value: False

The Weighted property determines the kind of formula used to calculate the Average points values.

Given this three points:

```
Point 0 :  x 1  y 4
Point 1 :  x 2  y 5
point 2 :  x 3  y 9
```

When Weighted property is False, the arithmetic average is used:

```
4 + 5 + 9 = 18
18 div 3 = 6   <------- 6 is the mean
```

When True, each value is multiplied by it's X coordinate:

```
( 1 * 4 ) + ( 2 * 5 ) + ( 3 * 9 ) = 41
```

Then the sum is divided by the sumatory of all X values:

```
41 div ( 1 + 2 + 3 ) = 6.833333333....
```

See Also
TTeeFunction.Period

FunctionType Property

Example

Applies to

TChartSeries component

Declaration

```
property FunctionType : TTeeFunction read FTeeFunction;
```

Description

The FunctionType property is used with TeeFunction components to define Period.

ChangeSeriesType Method

See also

Applies to
TChart class

Declaration

```
procedure ChangeSeriesType(Var
ASeries:TChartSeries;NewType:TChartSeriesClass);
```

Description
The ChangeSeriesType method changes the Series type for the Series of the given index.

See Also

TChart.AddSeries

The Chart Editor

Introduction

The Chart Editor is designed to help you quickly create and modify Charts. The Editor may be called at runtime using (TeeChart Pro only) using the EditChart method.

To get help on any Topic in the Chart Editor, select the button at the top righthand side of the Editor window and drag it onto the Topic in question. TeeChart Pro help will show you the runtime property or method associated with the feature.

Editor design

There are **2 principal sections** to the Chart editor, **Chart parameters** and the **Series parameters**, which are separated as 2 tabs of the Chart Editor.

Chart pages

You may define overall Chart display parameters at any time before, during or after adding Series to the Chart. Chart parameters are divided into the following sections:

Series page

You may add a mixture of different Series types to the Chart to define the specific Chart of your choice. Note here that you are **not limited to predefined Chart types**. Most Series types are compatible with other Series types on the same Chart, those Series types not available are greyed out. To add a new Series to a Chart select the Add button on this page which will display the Chart Gallery. Select the Series type of choice from the gallery and it will display on the Series page of the Chart Editor.

General Page

3D, Chart rectangle dimensions, margins, Zoom and Scroll, Print Preview and Export.

Axis Page

All Axes definition. Some parameters depend upon the Series associated with the axis, for example, Datetime depends on whether the Series data has datetime definition, this can be configured on the Series 'General' page of the Series concerned.

Titles Page

Teechart Title and Footer

Legend Page

Legend display. Formatted displays work in conjunction with the Chart Series. See also the 'General' page of the Series.

Panel Page

Chart Panel display properties. Colours, Bevels, Backimages, Colour Gradient and Border.

Paging Page

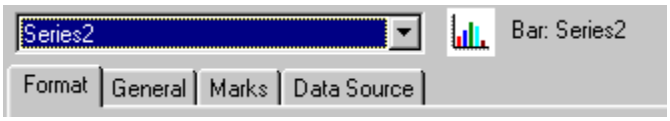
Definition of number of points per chart page. May be used to browse at design time, too, if your data is sourced from an ODBC datasource.

Walls Page

Left, Bottom and Backwall size and Colour definitions.

Series Pages

Series pages will contain parameters dependant on the series type concerned. The Combobox at the top of the Series tab page shows which series you are editing.



Format Page
Contains Series type specific parameters.
General Page
Series value format, Axis association
Marks Page
Series Mark format, text, frame and back colour and positioning.
Data Source Page
Access to Function definition and data sourcing for TTable, TQuery and TClientDataset

What's new !

Here is a list of some of the features new to version 4 of TeeChart.

3D improvements

- Improved 3D with rotation, elevation, scroll and zoom.
- 3D OpenGL rendering. **(TeeChart Pro only)**
- Create your own 3D rendering mechanism (for example VRML rendering, or DXF exporting).**(TeeChart Pro only)**
- New visual component TDraw3D for 3D generic drawing (non-chart related).**(TeeChart Pro only)**
- Faster drawing speed in most cases as now all drawings are performed directly to Windows GDI, bypassing the Delphi TCanvas when necessary.

New Series types (TeeChart Pro only)

- TRadarSeries (“spider” charts)
- TContourSeries (3D contouring)
- TPoint3DSeries (3D scatter with optional 3D lines)
- TBezierSeries (point smoothing)

All Chart components (TChart, TDBChart, TQRChart, TDecisionGraph...)

- A new Chart.DepthAxis to display labels (or Series titles) and ticks for the "Z" dimension in 3D mode.
- Retrieve Chart binary files from Internet URL addresses (LoadChartFromURL procedure). **(TeeChart Pro only)**
- New BackWall sub-component, with 3D depth.
- New Gradient filling styles (From / To Center, From / To Corner)
- New “OnBeforeDrawAxes” event. **(TeeChart Pro only)**
- Title and Foot accept now the Brush.Bitmap property to fill the background. **(TeeChart Pro only)**

Axes

- Multi-Line Axis Labels with and without rotation, at design-time and runtime.
- Unlimited multiple axis, connected to a single or to many Series. **(TeeChart Pro only)**
- All Axis (default and custom) can now be moved and stretched to any position **(TeeChart Pro only)**
- Centered Axis Grid lines

All Series

- Added "AddNullXY" method for all Series. **(TeeChart Pro only)**
- Shadow color 3D effect in Line, Area, Point, etc Series.
- Series HorizAxis and VertAxis properties can now show both axes at the same time. **(TeeChart Pro only)**

All Series Marks

- Custom positioning of Series Marks.
- Multi-line Marks text.

Fast-Line Series

- Fast-Line option to draw new added points (15000 points per second on a P166)

Pie Series

- Exploded Pie slices in both 2D and 3D, supporting rotation.
- Pie "Other" slice, grouping small slices into a single one.
- Brush.Bitmap for customized pattern filling.

Chart Legend

- Legend has now a Clicked function to return the Legend item index under the mouse.

- Multi-row Legends. **(TeeChart Pro only)**

TDBChart

- Series1.DataSource := DataSource1 for single-row (single-record) database charting.
- DBChart now uses the Delphi's TField "OnGetText" event. **(TeeChart Pro only)**

TQRChart (QuickReport Chart)

- New “OnPrint” event, to allow customizing the chart resolution.
- The “Frame” property now displays and prints. **(TeeChart Pro only)**

Line Series

- Line Series Height property, for 3D strip lines. **(TeeChart Pro only)**

Bar Series

- Bar and Horiz Bar Series can now be resized when zoomed (AutoBarSize property).
- BarBrush.Bitmap now can be used to fill the Bars.

Gantt Series

- Gantt points can be now customized using the OnGetPointerStyle event.

Shape Series

- New Shape styles (3D Cube, Pyramid, Invert.Pyramid, 2D Cross, Diag.Cross)
- Now Shapes origin can be expressed in axis values, while size expressed in pixels. **(TeeChart Pro only)**
- Brush.Bitmap for customized pattern filling.

Surface Series (TeeChart Pro only)

- Surface series easier to use with auto XZ grid. Allow specific non-sorted XYZ points
- Legend now shows the Surface Palette
- Surface series not obligued to add all points (missing points are considered nulls).
- Surface holes using the AddNull method.
- Multiple-surfaces in the same chart.

Editor Dialogs

- Chart Editor now uses the TOpenPictureDialog Delphi's dialog.
- "Edit..." button for TeeFunctions at Editor dialog to modify the Function.Period. **(TeeChart Pro only)**
- New components: TChartEditor, TChartPreviewer to configure and show the Editor and Preview dialogs. **(TeeChart Pro only)**
- Several new options at editors to allow to use the new features.

Functions

- TTrendFunction is now much faster. **(TeeChart Pro only)**
- New TStdDeviationFunction. **(TeeChart Pro only)**

Other

- TChartScrollBar now installed by default. (It was an option in 3.0 version) **(TeeChart Pro only)**
- TeeComander component: A toolbar panel with buttons for 3D rotation, scroll, OpenGL, etc. **(TeeChart Pro only)**
- New global boolean constant: TeeDrawAxesBeforeSeries (by default True). **(TeeChart Pro only)**
- TVolumeSeries has now it's own editor dialog. **(TeeChart Pro only)**
- Big, big, big code re-organization and stream-lining (in major part because the new 3D features)

Introduction

TeeChart version 4 contains many exciting new features including a new 3D Canvas both for the Chart and Custom drawing.

The 3D Canvas is available as the native TeeChart orthogonal view, by Native Windows graphics or with OpenGL. The 3D Canvas offers up many new possibilities for the display of impressive presentation Charts. The 3D Custom draw methods support the drawing of 3D objects anywhere on the Chart canvas.

There are many new enhancements and extensions to both Chart and data Series types. TeeChart remains 100% VCL code. Many Series types have been enhanced for the Standard and Pro versions and the Pro version has several brand new Series types including Bezier, Contour, Radar and Point3D . Included in the Standard version is the new 'explodable' Pie Series which also has new 'Slice grouping' functionality.

Still inherent is the flexibility to mix and match different Series across different Charts. New Function definitions can be applied to any group of your data Series across Charts.

Support for JPEG file export has been built into the new version, a new Backwall has been added to the Chart as have multiline Axis labels and innumerable low level functionality enhancements.

Series can be connected to Tables, Queries and RemoteDatasets residing on different Forms or DataModules. You can create ActiveX Forms containing Chart components and deploy them on the WWW using Delphi's connectivity functionality.

TChart is seamlessly integrated with QuickReport with a custom built TChart component on the QuickReport palette.

TeeChart online help has now grown to take in the user manual, hopefully making it easier for you to get to the information you need. The help Contents file lists the information classifications included in this helpfile.

See What's New for a list of all new features in TeeChart version 4

We hope you enjoy working with TeeChart 4 !

About TeeChart

TeeChart Pro order information License and disclaimer

TeeChart Charting components have been written in Delphi by David Berneda. They are 100% Delphi Native Visual Component Library compliant.

For those of you familiar with TeeChart you may wish to look at What's new for a listing of many of the new features of version 4.

TeeChart Standard version 4

Runtime TeeChart libraries are included as part of Delphi versions 3, 4 and 5. They include many compiled units of the Chart components and all compiled standard Series types. TeeChart v4 has a design time Chart Editor, extensive coded examples and demos, online help with user guide.

Features of TeeChart 4 Standard version

TeeChart Pro version 4

TeeChart Pro is a comprehensive Charting tool aimed at those developers wishing to program with or make use of extended TeeChart functionality. TeeChart Pro includes the option to purchase **100% source code**.

Extended features of TeeChart Pro

With **TeeChart Pro** you define the limits. Use the developer guide to help you create hot-spots for drilldown on your Charts or customise the TeeChart code to create your own Series types and functions and permanently add them to the TeeChart Gallery.

TeeChart User Guide

The TeeChart User Guide provides a backgrounder and an indepth look at most commonly used and some advanced Chart design techniques.

Getting Started

This section should bring you up to speed if you are trying out TeeChart version 4 for the first time. Building a Chart from scratch is very quick with TeeChart so we recommend it worth your while to run through the included examples in your Delphi IDE.

Topics in this section:

Using TeeChart on-line help

Selecting a TeeChart component

Whilst designing your project using a TeeChart component you may select the component from the component palette or on the form where you have placed it and key **F1**. Context sensitive help relating to that component will appear.

Selecting a TeeChart property, method or event

Highlight a property or event in Object Inspector and key **F1** to obtain help for that item. Alternatively write the word for which you would like some help in your code. Select the word by placing and clicking the cursor on that word and key **F1**. If help is available, TeeChart context sensitive help will access directly a description relating to that functionality.

Help Contents page

The Delphi help Contents page includes a branch dedicated to TeeChart. You can follow the headings there to take you to particular reference section.

Using the TeeChart Wizard

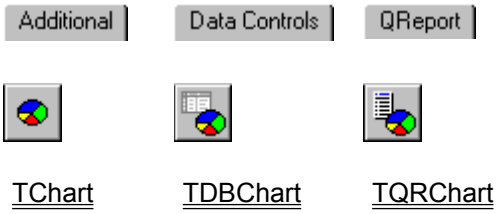
The TeeChart Wizard is only available for use with 32bit Delphi.

To create a Chart using TeeChart Wizard select the **File** menu in Delphi and **New**. Choose **Business** from the tab selector in the **New Items** window. You will see the TeeChart Wizard icon.

Simply double-click the icon to start creating Charts! The Wizard will lead you through the steps necessary to create your own non-database or database aware Chart.

Which TeeChart component should I use?

There are three TeeChart icons in the component palette.



TQRChart, is a component especially tailored for use with QuSoft's QuickReport - See the chapter Adding a Chart to QuickReport.TQRChart is descendant of TDBChart.

The 2 components, **TChart** and **TDBChart** are the basic building blocks of all TeeChart Charts.

If you wish to create the data Series for your Chart manually, or via a coded procedure then use the TChart component. If you wish to source your graph data from a table or SQL query then use the TDBChart component. You may, of course, use TDBChart in all cases but it will result in a larger compiled size of your code which would be inefficient if you are not using a datasource (explanation follows).

Once an initial data Series is defined, both Chart components, TChart and TDBChart support the use of another data Series as a data source.

- So why two kinds of Chart components?

- couldn't we achieve this functionality via just one component?

Answer:

Projects compiled with using only TChart components, **don't need** the Borland Database Engine DLL's (BDE).

Internally, both derive from the TCustomChart component, which is responsible for most Charting capabilities.

In Delphi 4 and 5 TClientDataset can be accessed without the need for BDE.

Creating a new Chart with *TChart* or *TDBChart* component

Many of the steps necessary for defining a TChart or TDBChart are the same. This section describes the common steps needed to start defining either of the two types of Chart.

Create a new Form.

Create a new form and place a TChart or TDBChart component on it (drag it across from the component palette).

Drag the corners of the new TChart or TDBChart component out to a size that helps you visualise the contents of the new Chart as you define it. Later you can adjust the size of the Chart to suit your needs.

Edit the new Chart

Position the mousepointer over the new Chart and press the right mousebutton. A menu appears that includes the 'Edit Chart' option.

Fig. 1. Using the right mouse button on the Chart will call the Chart options menu.

Select the **Edit Chart…** option to edit the Chart and define and populate its data Series.

Go to the next step.

The Chart editor

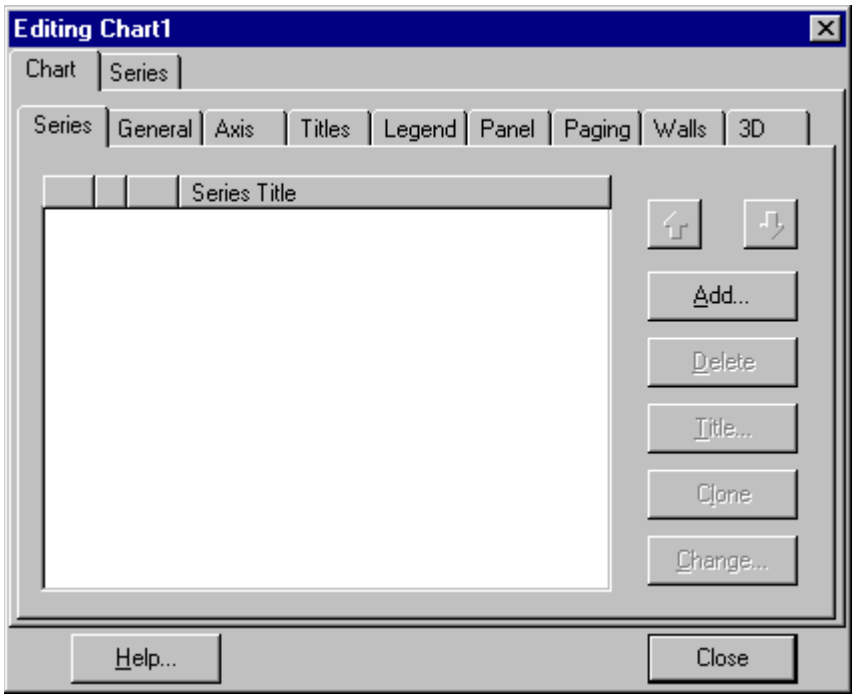


Fig. 2. The Chart Editor screen

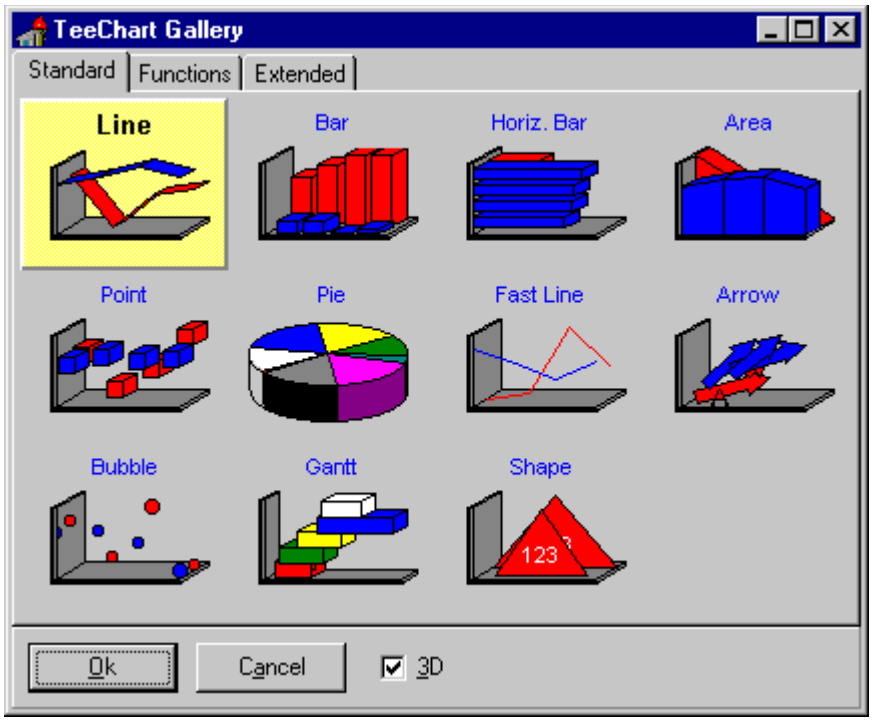
The Chart page (1st page) of the Chart editor contains definition information for the Chart. It includes sections to define general and other more specific, Chart parameters. Some parameters won't apply until you have some data Series defined in the Chart. Try modifying a parameter, the Title for example, and you will see it update in real time on the Chart.

To see some data Charted we need to create a data Series...

Add a data Series

Press the **Add** button in the **Series** tab section of the Chart page. TeeChart will show you a gallery of Series types. Select one to add to your Chart, you can change its type later if you decide that you would prefer to visualise your data in a different way.

Fig. 3.
The Chart Gallery

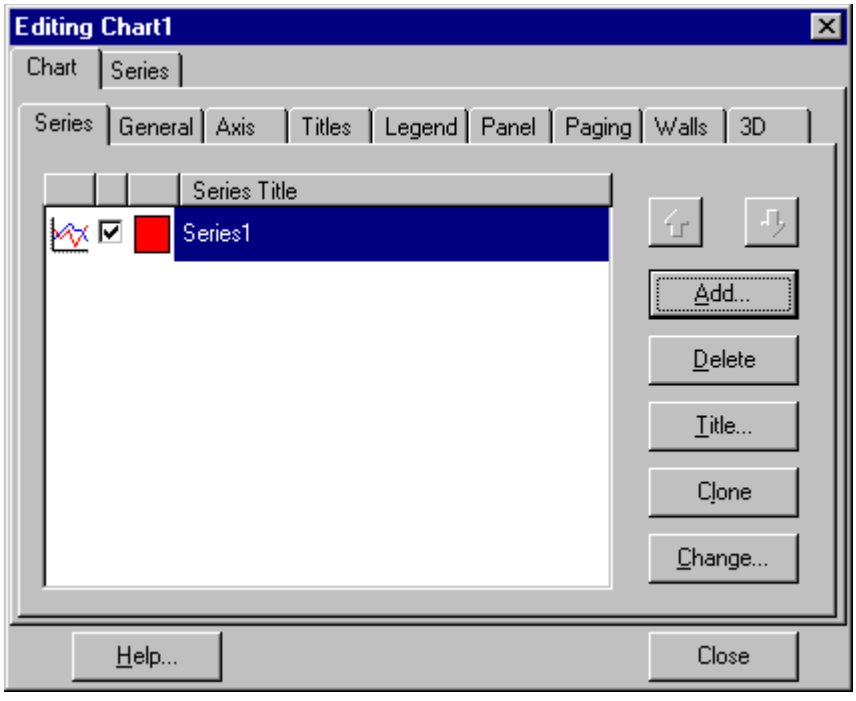


Select a Series type by mouse or with arrow keys and press 'OK'. Double-clicking on the Series type will achieve the same result. The Extended Series page is only available in the Gallery with TeeChart Pro.

The Series type is automatically added to your Chart. In the Chart editor you will see a new configuration tab added for the new Series.

Suppose we had selected a line Series. The editor will appear as follows:

Fig. 4.
New Series added to Chart



TeeChart has added the Series to the Chart. It has also added some random values so that you can

visualise, in the Chart, the Series appearance at design-time to easily follow any changes you are making.

Press the **F9** key to compile your project. The project should compile to show you an empty Chart. The random values don't work at runtime so the next step is to go back to the Chart Editor and add a data source or write your own code to add data values.

Edit the Series

Selecting the Series tab allows you to edit your Series. The next step is to add data to the Series. The steps necessary to include the data for the Series vary slightly between a TChart component and a TDBChart component. See the following 2 *sections:*

Configuring data Series in a TChart.

and

Configuring data Series in a TDBChart.

Configuring data Series in a *TChart*

We have a TChart on the form and have added a Series to it. We are ready to populate the Series.

Let's type some Pascal code to add points values programmatically.

We'll see later how to create a database-aware Chart with automatic record retrieval.

Example Pie-Series

Suppose the Series we added was a Pie Series. We could populate the Series in the following way. For the following code to work we should leave the Series name as its default of *Series1*.

Place a TButton on your Form and go to the OnClick event .

Copy the following code at the Button1.OnClick event:

```
With Series1 do
Begin
  Add( 40, 'Pencil' , clRed ) ;
  Add( 60, 'Paper', clBlue ) ;
  Add( 30, 'Ribbon', clGreen ) ;
end;
```

A description of the Add method and other available methods and properties is available at design time via the context sensitive help included with TeeChart. Place the cursor on the word *Add* in your code and press **F1** for a full description of the method.

Return to your code and press **F9** again to run the project.

Press **Button1** to see the Pie Chart appear - the code works!

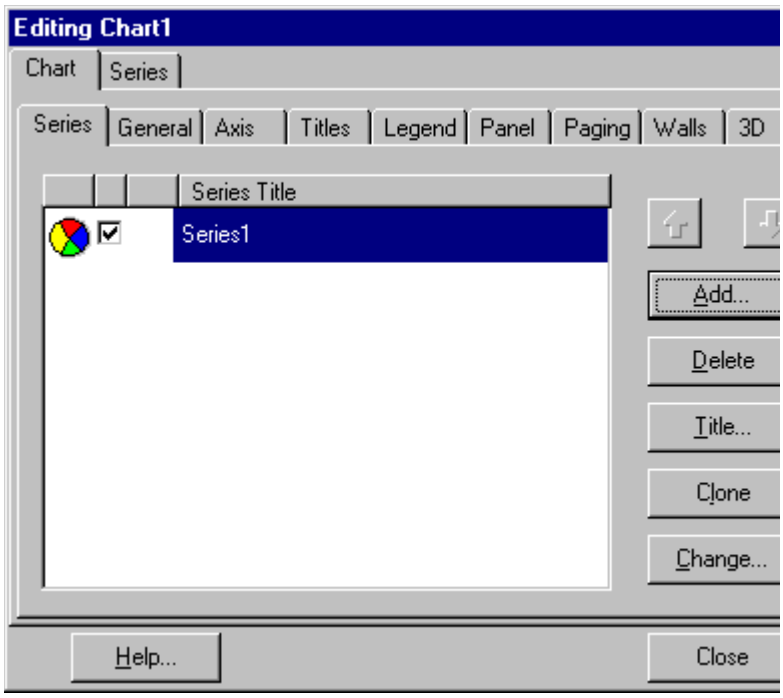
Editing the Series

Close the program to go back to Delphi's IDE.

In design mode, all Chart and Series properties are accessible through the Delphi Object Inspector or by **right-clicking** the Chart and selecting the 'Edit Chart' option or double-clicking on the part of the Chart that you wish to edit. Here we'll use the Chart editor to edit our new Chart and Series. **Fig.5.** Shows the first editor screen.

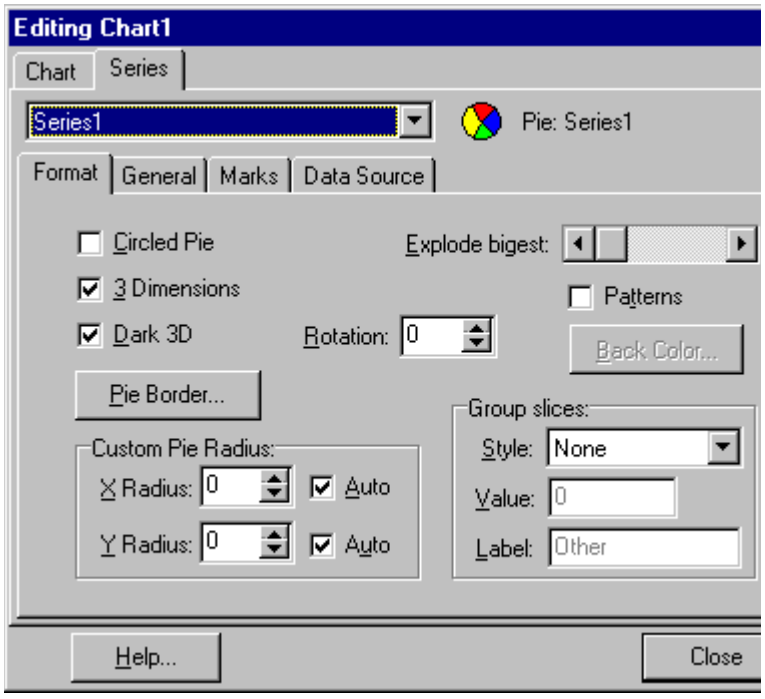
To edit features of the Series we can double-click on the Series in the list or highlight the Series and select **Edit** or directly select the tab for the Series - each technique will take us to the editor for the Series.

Fig. 5.
The Chart Editor screen showing the new Pie Series



Try changing some properties of the Pie Series; you will see the results update automatically on the Chart. There is no cancel to undo changes but most parameters are easily toggled. Remember, in design time the TChart component hasn't yet run your code so it shows the randomly generated set of data. You will need to start your project to see how the new parameters show up with your own data.

Fig. 6.
The Chart Editor Series page showing editor tab for our Pie Series



Many changes can be made visually in the editor, or try modifying some parameters in Object Inspector, or modifying a property with your own code. For more advanced projects, you'll most likely find yourself typing some Object Pascal code.

For a recap and further look at adding data to a TChart via another example, see How To Create Charts

with manually inserted values

Now we'll take a closer look at adding data to the TDBChart component.

Configuring data Series in a TDBChart.

Configuring data Series in a *TDBChart*

Prior to using any database facilities, you should have correctly installed the Borland Database Engine (or equivalent) and the Database Delphi VCL components.

A Step by step guide to creating a database aware Chart....

- Place a TTable component and point it to the **"DBDEMOS"** database and the **"ANIMALS.DBF"** dBase table.
- Place a TDatasource component and set its **DataSet** property to **Table1**.
- Place a TDBGrid and set the DataSource property to **DataSource1**.
- Set the Table1.Active property to **TRUE** to see the Grid filled with table data.
- Place a TDBChart component onto a Form.
- Select and install a Pie Series as described in section The Chart Editor.
- Go to the **Series page** of the **Chart editor** by either double-clicking on the Series name, highlighting the Series and selecting **Edit**, or by selecting the Series tab.
- Select your Series from the Series listbox and go to the datasource tab.

From the listbox select your data source type.

There are 4 types of data source available. The DataSource style is a component property that may be one of the following:

'No data' if points are being added **programmatically** (at source code).

or...

'Random values' draw your Chart Series with random values.

or…

'A function' which may be:

One or more other Series (like LineSeries1, BarSeries2, etc.) which could come from this Chart or from another Chart in your project. **A *function*** may work with a combination of one or more other Series and an algorithmic function (min, max, average, etc.).

or..

A **'Dataset'** which may be a:

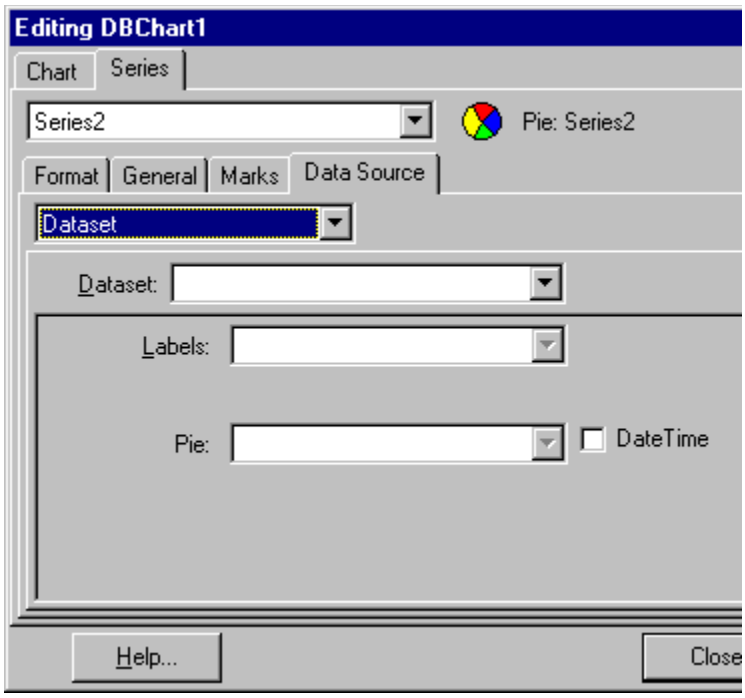
TTable, TQuery, TClientDataset or any other TDataset component. (like Table1, TQuery1, etc.)

Adding a dataset

For our example, let's choose the 4th option from the previous section, **'Dataset'** style. In our TDBChart we'll include data from the table that we've already added to the form.

You need to select the listbox option **A dataset** to map this Series to the table that you have included in the project. As you make the selection you will notice new page options appear to define the dataset for the pie Series. Your editor should look like **Fig. 7**.

Fig. 7.
The Chart Editor screen showing the Data source tab for our pie Series



All you need to define a simple dataset for the Series lies here. You have already included a *table* and *datasource* in your form, and set the table to *active*. But the options here offer you the flexibility to define the dataSeries in varying ways:

Click on the down-arrow for the **'Dataset:'** combobox.

If you have an activated table in the form it will be

highlighted here.

Select your table.

Define the values that will be plotted in the Chart:

As we have defined a Pie Series we are presented with pie specific configuration options. Each Series has parameters that vary, but only slightly, in definition from the example below.

The **Pie** field is mandatory, and must contain a **valid numeric** field or a string field with values that can be converted to numbers.

In our example, choose the **Weight** field.

(The *ANIMALS* table is a collection of animal names with weight and size information.)

You can optionally select an **Labels** field to draw the corresponding **string** for each point on the pie.

In our example, choose the **Name** field.

This label will be used to draw the Pie **Legend**.

Note

Some Chart Series types have more than one figure to represent a point. Values that are plotted by date or time are an example of **X Values** Series. Not all date-time values should be equally spaced. The **X Value** is the specific **horizontal position** for each **Y value**. Later try deleting the pie Series and adding a Series of a different type, - ie. LineSeries -, you will see the option to apply X and Y values appear.

(We don't have X Values in a Pie Series.)

Now click the **"OK"** button.

The TDBChart component will show a **Pie** Chart and a **Legend**.

You are seeing **exactly** the same Chart you saw when you pressed **F9** to run the project.

For a recap and further look at adding data to a TDBChart via another example, see How To Create Data Aware Charts

Modifying Chart data

Chart data may be accessed via the the Point Valueindex of each Series' points.

Deleting points:

Use the Series Delete method:

```
{This will delete the 5# point from LineSeries1}
LineSeries1.Delete( 5 );
```

Modifying points:

Use the Series XValue, YValue properties:

```
LineSeries1.YValue[3] := 27.1 ;
{In Bubble Series}
BubbleSeries1.RadiusValues.Value[ 8 ] := 8.1 ;
{In Pie Series}
PieSeries1.PieValues.Value[ 3 ] := 111 ;
```

Please refer to each specific Series type to see its specific data related methods.

DateTime point coordinates:

TeeChart Series accept DateTime values.

```
{First, you need to set the DateTime property to True in the desired X
  and/or Y values list.}
LineSeries1.XValues.DateTime := True ;
{Second, use the same above described methods, but give the values as Date,
  Time or DateTime values}
LineSeries1.AddXY( EncodeDate( 1996 , 1 , 23 ) , 25.4 , 'Barcelona' ,
  clGreen );
```

TeeChart demos

The Getting Started section of the TeeChart User Guide explains the most basic TeeChart procedures. Another useful place to look for help are the examples in the demo files.

Location of TeeChart demos

Open in **DEMOS\TEECHART** folder the **TEEDEMO.DPR** Delphi Project. The DEMOS Folder is located below your Borland installation Folder.

There are many included forms in this demo.

Each form shows a specific TeeChart feature or Series type.

Many Forms contain Pascal source code to show event handling or interactive Charting.

Let's see the **ScrollForm** demo form. Click on Project Manager and search for **USCROLL** unit or **ScrollForm** form. This form shows a basic TChart component with one **TLineSeries** component.

Note:

In this example we've named the Series 'LinesSeries1' to clearly identify it in the code. It's not necessary though to name the Series according to Series type as the flexibility of TeeChart's components offers the ability to change type at any time - thus Series names by default are generic, Series1, Series2, etc..

Click on **Form.OnCreate** or switch to source code to see the following:

```
procedure TScrollForm.FormCreate(Sender: TObject);
begin
 FillDemoPoints;
end;
procedure TScrollForm.FillDemoPoints;
var t:Longint;
begin
 { fill the LineSeries with some random data }
 LineSeries1.Clear; { <-- this removes all points from LineSeries1 }
 { let's add 60 minutes from 12:00 to 12:59 }
 for t:= 0 to 59 do
   AddXY( EncodeTime( 12, t, 0,0),Random(100),",clRed );
 { let's add 60 more minutes from 13:00 to 13:59 }
 for t:= 0 to 59 do
   AddXY( EncodeTime( 13, t, 0,0),Random(100),",clRed);
end;
```

This code will **Clear** LineSeries and plot two hours of **random** values from 0 to 100.

Because it is placed on the **Form.OnCreate** event, each time the form is shown, this code is executed.

Double click the **Button1** component to see its associated **OnClick** code:

```
procedure TScrollForm.Button1Click(Sender: TObject);
var h,m,s,msec:word;
begin
 if CheckBox1.Checked then { If VERTICAL SCROLL }
   DecodeTime( LineSeries1.YValues.Last , h, m, s, msec)
 else
   DecodeTime( LineSeries1.XValues.Last , h, m, s, msec);
 { add a new random point to the Series (one more minute) }
 inc(m);
 if m=60 then
 begin
  m:=0;
  inc(h);
 end;
 AddXY( EncodeTime( h, m, s, msec), Random(100),", clYellow );
```

```
end;
```

This code increments the **last added point** Time by one more minute.

The incremented Time value is added to the LineSeries, thus giving a new point.

The next source code is the demo purpose:

How to scroll the Horizontal Axis as new Points are added to the end ?

The easy way is to *type* the following code at the **Series.OnAfterAdd** event. This event is called *each time* a new point is **added** to the Series.

In this example, we **scale the bottom horizontal Chart axis** to show 55 minutes prior to the last point and 5 minutes after the last point.

```
{ This is the event we need to arrange Axis scale as new points are added. }
procedure TScrollForm.LineSeries1AfterAdd(Sender: TChartSeries;
 ValueIndex: Longint);
begin
 { If VERTICAL SCROLL }
 if CheckBox1.Checked then
 begin
  With Sender.GetVertAxis do { <-- with the
            Vertical Axis... }
  Begin
   Automatic := False;    { <-- we don't want
         automatic scaling }
{ In this example, we will set the Axis Minimum and Maximum values to
show One Hour of data ending at last point Time plus 5 minutes }
   Minimum := 0;
   Maximum := Sender.YValues.MaxValue +
         DateTimeStep[ dtFiveMinutes ];
   Minimum := Maximum - DateTimeStep[ dtOneHour ];
  end;
 end
```

Many goals can be achieved by manually setting the Axis properties.

Each Chart component has four axis: **LeftAxis, TopAxis, RightAxis and BottomAxis**. **Each** Axis has a boolean "**Automatic**" property that defaults to **True**, meaning TeeChart will **always calculate** the **Minimum** and **Maximum** values for each Axis.

The **Minimum** and **Maximum** properties are of type **double**, allowing float numbers or **DateTime** representations like "Date", "Time", "Now" or any other valid datetime value.

We have seen a specific TeeChart feature in detail.

Now, compile and execute this project *and look at each sample form*.

They should give you ideas that you can apply to your projects while you learn about TeeChart features.

The **TEECHART.HLP** file describes each component property in detail, with some source code examples mostly taken from the **TEEDEMO** project.

We suggest that you to experiment with each different sample form and create new small projects to test TeeChart with your *real data.*

TChart component

The TChart component is the basic building block for ‘non-database-aware’ Charts. Select the TChart component from the Delphi palette and simply drag it onto your form to include a Chart in your application.

See TChart Component for Class definition.

You can use the Chart Editor to define display characteritics for the Chart and to add new Data Series. Alternatively all properties can be set and changed at runtime using Chart properties and methods.

Data Series for the TChart component must be populated by code. See Configuring data Series in a TChart for an introduction.

For an indepth look at how to create Charts with TeeChart see the TeeChart User Guide.

TDBChart component

TDBChart derives from TChart and inherits all its functionality. When a Chart Series is connected to a TDBChart component, TDBChart looks in the Series DataSource property.

The datasource for the Series is defined by the Series definition - not by the Chart - Thus multiple Series in a TDBChart could access different data sources. However, if those Series are not associated, via ParentChart, with a TDBChart then the option to define a Series database-datasource won't be available.

Creating the Dataset

TeeChart Charts will connect with 3 different types of Dataset.

TTable

TQuery

TClientDataset

The minimum that requires to be set in each case is:

The name of the database,

For the table the TableName of a table

or

in the case of the query a valid SQL string.

or

in the case of a ClientDataset Define source (right button on TClientDataset object in form).

Remember to activate the Dataset by setting the Active property to True.

Connecting your Series to a database Dataset

In 'Getting Started' you saw how to connect a data Series to a database dataset. Let's recap here on the key components.

When you select your new **Series** in the **Chart Editor Series page** you will see the tab option for **Data Source**. If you want your data Series to be connected to a new dataset then you should select **Dataset** from the drop down combo listbox. A new selector box will appear with the options for definition of the new dataset.

The exact contents of the page will change depending on the Series type you have chosen. The parameters you set here modify the properties in the Series definition which vary between Series type. The following table shows the possible options for normal Series types (function Series are different).

SERIES TYPE	DATASOURCE PROPERTIES
Basic	
Line	XValues, YValues, XLabel
Fast Line	XValues, YValues, XLabel
Bar	XValues, YValues (called Bar), XLabel
Area	XValues, YValues, XLabel
Point	Xvalues, YValues, XLabel
Pie	PieValues, XLabel
Arrow	StartXValues, StartYValues, XLabel, EndXValues, EndYValues
Bubble	Xvalues, YValues, XLabel, RadiusValues

Gantt	StartValues, EndValues, AY (Y axis level), AXLabel (Label optionally shown on Y-axis or as mark)
Shape	X0 (Top), Y0 (Bottom), X1 (Left), Y1 (Right)

Extended

Bezier	XValues, YValues, XLabels
Candle	OpenValues, CloseValues, HighValues, LowValues, DateValues
Contour	XValues, YValues, ZValues, Labels
Error Bar	XValues, YValues, XLabel, ErrorValues
Polar	XValues, YValues
3D Surface	XValues, YValues, ZValues
Volume	XValues, YValues (VolumeValues), XLabel

Coding your datasource

You may populate your Chart at runtime by coding which Series to add to the Chart and defining the fields of those Series.

It assumes you have a table on your form, *Table1* with fields *Name* and *Amount*.

```
Var
MySeries,Ave:TLineSeries;
MySeries:=TLineSeries.Create( Self );
With MySeries do
begin
 ParentChart:=DBChart1;
 DataSource:=Table1;
 XLabelsSource:='Name';
 YValues.ValueSource:= 'Amount';
 CheckDatasource;
end;
```

TChartSeries

The TChartSeries component is the ancestor of all Series types. When referencing the properties of the Series type with which you are working check, in addition to the properties listed with that Series, the properties of the TChartSeries to get a full list of properties common to all Series.

Topics included in this section:

Line and Fast Line

Bar

Horizontal Bar

Area

Point

Pie

Arrow

Bubble

Gantt

Shape

Combining Series

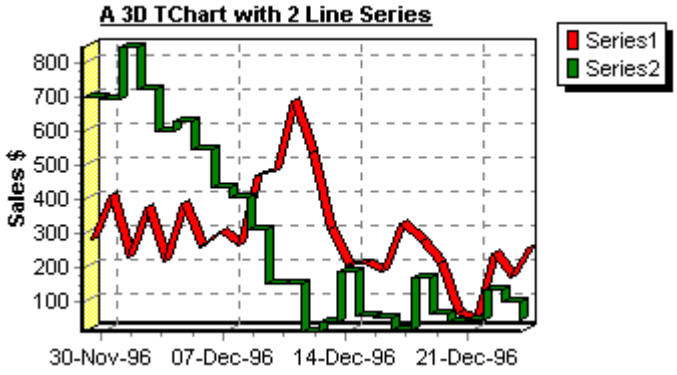
Line Series

There are 2 line Series types available, Line and Fast Line. Which one should be used ? Fast line is just what its name describes - it is fast. It is distinct from Line because to achieve speed - speed to add new points to the Series - the price paid is that it foregoes some properties that the Line Series has. See the section on Fast Line for a description of those differences.

Line

See TLineSeries for a full list of properties and methods

Fig. 1.
A 3D Line Series showing one Series with the **stairs** property set to true. The stairs can be inverted.



Fast line

See TFastLineSeries for a full list of properties and methods

This line Series draws only at 2 Dimensions but draws very quickly - performance will depend on your hardware - The Series type was originally conceived to tackle high volume requirements of technical and financial applications but serves well for any dataset of very high point volumes.

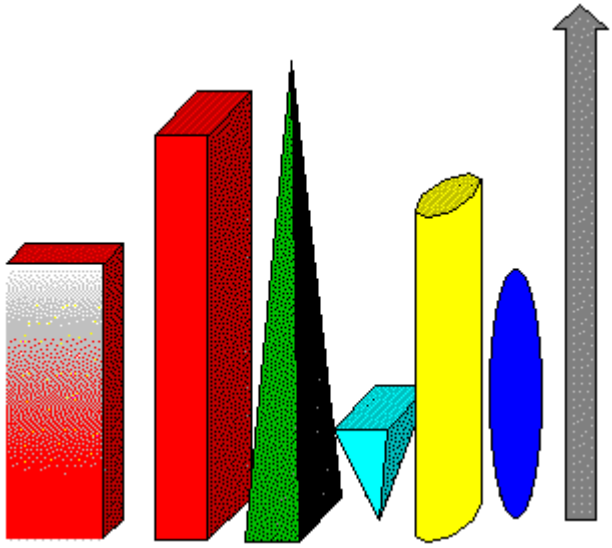
If, when using Fast Line you know approximately how many data points will be added to your Series you could use the global variable TeeDefaultCapacity to prepare memory. For example, assuming a data Series of 10,000 you should set TeeDefaultCapacity to 10,000. This will, in one pass, allocate memory to populate the whole Series replacing the need to incrementally allocate memory which costs more time.

Bar

See TBarSeries for a full list of properties and methods

The bar Series in 2 or 3 dimensions doesn't have to be represented by a rectangular bar -

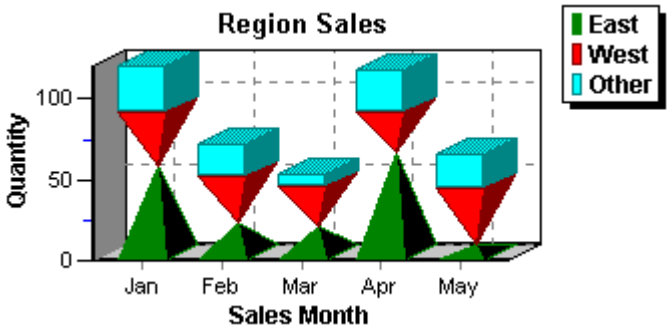
Fig. 2.
Choose a barstyle for your Chart Series or 'mix and match' to suit your needs.



Example bar Series configuration

Mixing bar Series styles may be useful for some applications. Below is an example stacked bar Chart.

Fig. 3.
Mixed bartypes



Steps taken to build this demo Chart and populate it with data are as follows:

1. Drag a Chart component onto your form and drag it out to the size you want.

2. Click the right mouse button over the Chart and select editor from the menu.

Add 3 **bar** Series and give them individual titles using **Title** on the first editor screen (titles in our example 'East' 'West' and 'Other').

Select **Multi Bar** and set as **stacked** in the **Format** page of any one of the 3 Series (The change will apply to all 3 Series).

Add a title using the entry tab for '**Titles**' in the Chart page of the editor.

Go to the axis tab. Select Left axis and add the title. We allowed the axis to automatically size itself but took **minor ticks** off on the bottom axis.

Each Series has a different shape defined for its bars. We define the shape individually in the configuration page of each barSeries. The first tab, **Format**, has the option for **Style**.

You could code the style with:

```
East.BarStyle:= bsPyramid ;
West.BarStyle:= bsInvPyramid ;
Other.BarStyle:= bsRectangle ;
```

Lookup BarStyle in the TeeChart online help to get the full list of options.

For this Chart it would be necessary to control the **Series order** to achieve the correct effect. This can be done in the Chart Editor at design time by selecting the Series in the Series list of the Chart page and using the arrows to change the order. At runtime we can achieve the same with the following code:

Drawing order is always based on the order for which the Series are assigned to ParentChart.

The order can be altered at run-time via the Serieslist property:

```
Chart1.SeriesList[ 0 ] := East;
Chart1.SeriesList[ 1 ] := West;
Chart1.SeriesList[ 2] := Other;
```

We added some random values to populate this demo. We could launch the code at form creation or put a button on the form to run the code.

```
procedure OurForm.Button2Click(Sender: TObject);
var t:integer;
begin
Randomize;
with East do
for t:=1 to 12 do AddY( Random(70), ShortMonthNames[t],clTeeColor);
with West do
for t:=1 to 12 do AddY( Random(70), ShortMonthNames[t],clTeeColor);
with Other do
for t:=1 to 12 do AddY( Random(30), ShortMonthNames[t],clTeeColor);
end;
```

We populated the data for 12 months but are showing only 5 months. The Paging tab in the Chart page of the editor has the definition of points per page.

Bar Series display

The **Multi Bar** parameter set in the Format tab of the Series page for one of the bar Series sets the display alignment of the bars for each bar Series in the Chart. - It is not necessary to go to each bar Series to change the parameter.

The following figure shows different formats to display bar Series. Each Chart has the same information only displayed in a different way. The Stacked 100% Series display doesn't represent the actual values but rather the relative value of each element of the Series to a total of 100%.

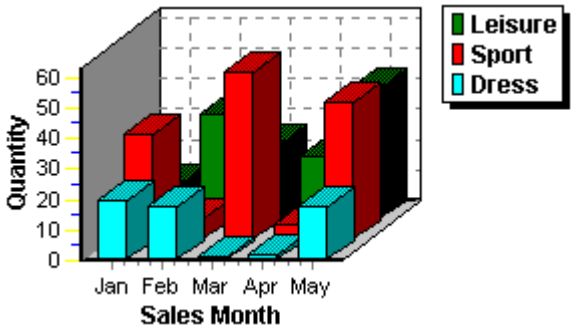
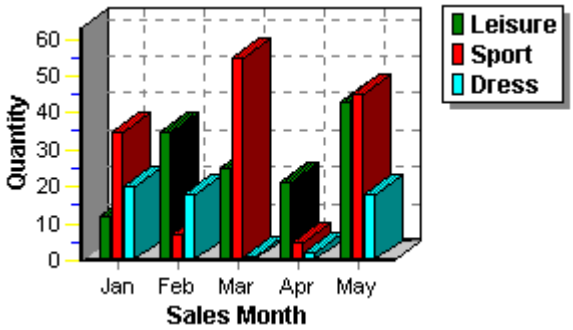


Fig. 4. a
A 3D Bar Series showing four methods to display the same information.

Stacked property

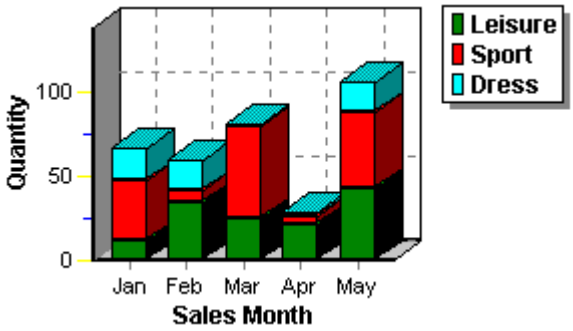
None

Fig. 4. b



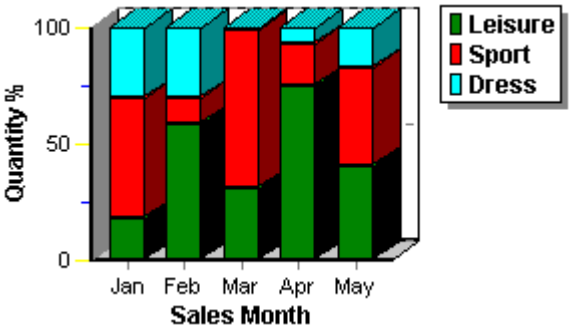
Side

Fig. 4. c



Stacked

Fig. 4. d



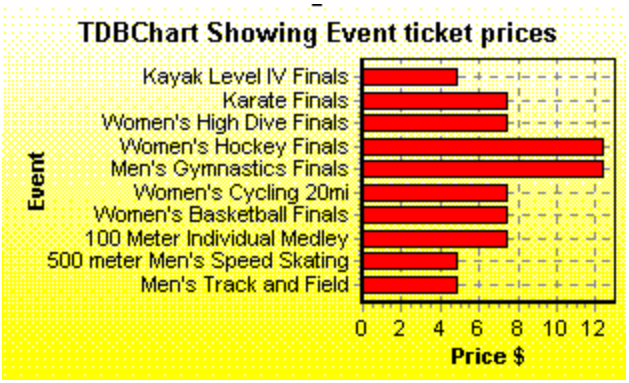
Stacked
100%

Horizontal bar

See THorizBarSeries for a full list of properties and methods

The horizontal bar Series shares the same properties as the bar Series. Apart from any aesthetic requirement, one occurrence of the need to use a horizontal bar Series may be to adequately display long axis labels which are best read horizontally.

Fig. 1. 2D Horizontal bar

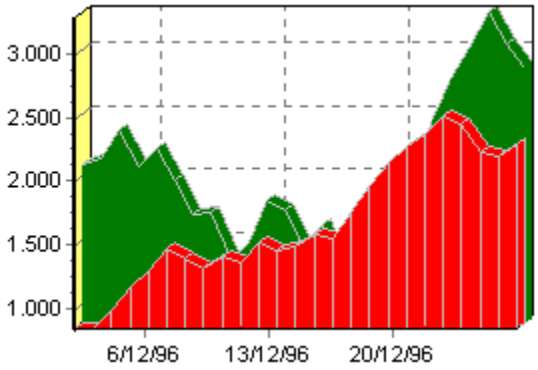


Area

See TAreaSeries for a full list of properties and methods

An Area Series has similar characteristics to a line Series - filled -

Fig. 5.
3D Area Series

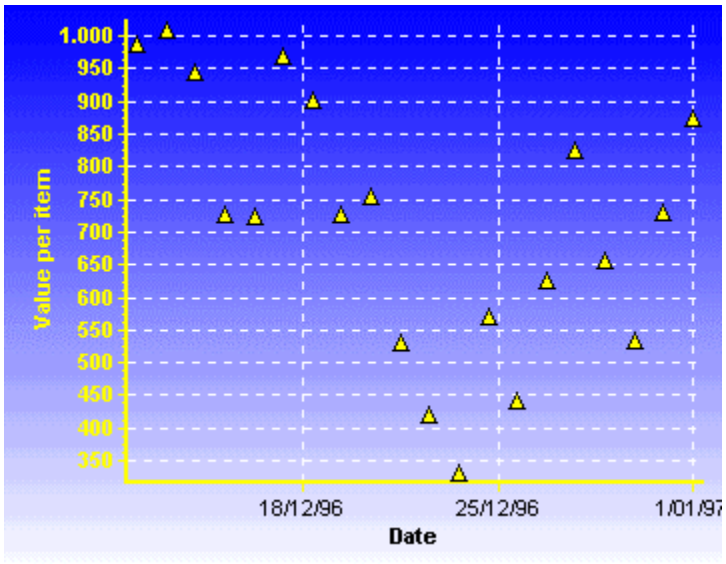


Point

See TPointSeries for a full list of properties and methods

A Point Series is similar in definition to a Line Series without the line.

Fig. 6.
2D
Horizontal
bar

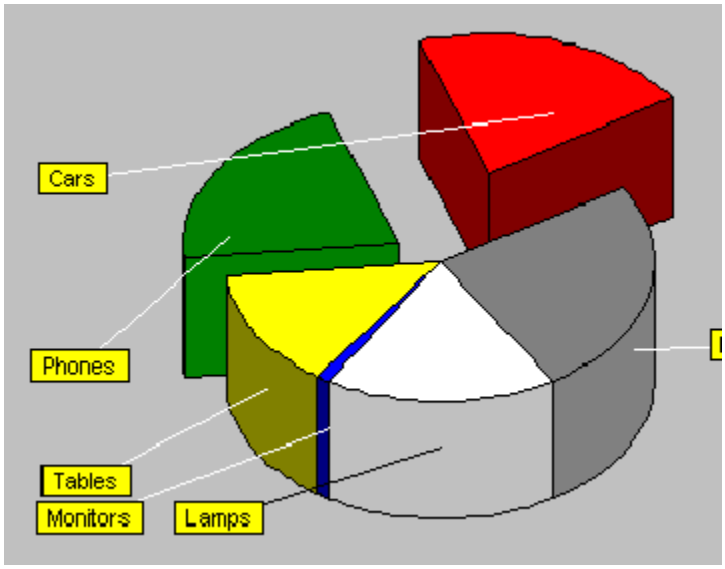


Pie

See TPieSeries for a full list of properties and methods

A Pie Series is unique in not needing any axis. It is possible to mix a Pie Series in a Chart with another Series that requires an axis.

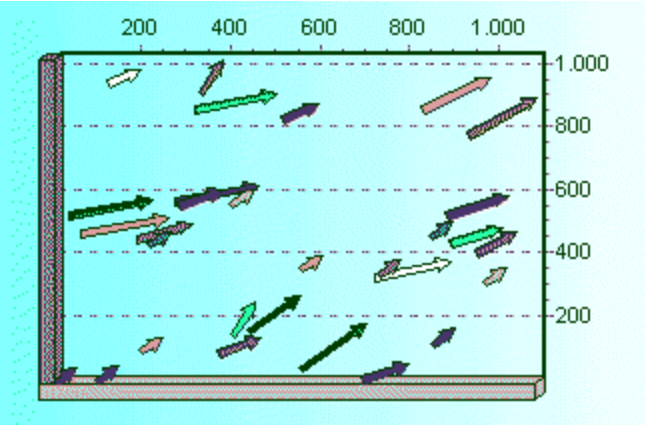
Fig. 7.
3D Pie



Arrow

See TArrowSeries for a full list of properties and methods

Fig. 8.
3D Arrow Series



The arrow Series is useful for displaying start and end points of many individual events.

Bubble

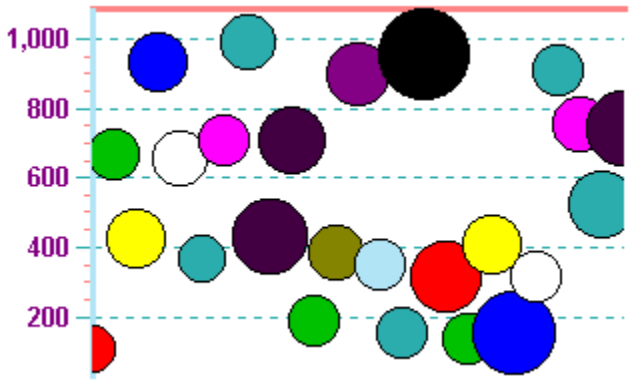
See TBubbleSeries for a full list of properties and methods

The Bubble Series has 3 configurable parameters that define the value of the data in your Series.

Xvalues, YValues, RadiusValues

The bubble Series is useful for showing importance weighting. For example, comparing high volume selling product that, by income, doesn't bring in a revenue of the scale of another low volume seller. We can see at a glance, literally, that big bubbles are important !

Fig. 9.
2D Bubble Chart



Shapes

Bubble Series can be configured in variable shapes, triangles, etc..

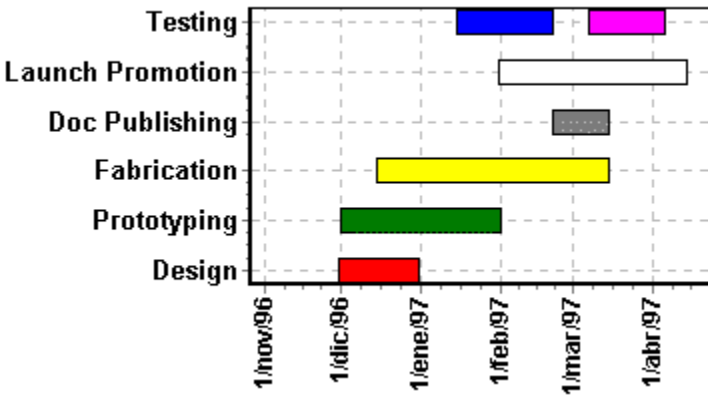
Gantt

See TGanttSeries for a full list of properties and methods

Use the Gantt Chart as a planner or to track progress of a project or Series of activities.

The Gantt Series draws bars that have start and end values which may be of datetime format. You may define a Y axis value for the vertical position of the bar and you may define 'next bar' to draw connection lines between the bars.

Fig. 10.
2D Chart with a Gantt Series.



How to add Gantt bars manually

Use the AddGantt or AddGanttColor methods.

Example:

```
GanttSeries1.AddGantt( EncodeDate( 1997, 1, 1 ),
            EncodeDate( 1997, 1, 31 ),
            0,
            'Programming' );
```

Or...

```
GanttSeries1.AddGanttColor(EncodeDate( 1997,1,1 ),
             EncodeDate( 1997,1,31),
              0, 'Programming',
              clGreen );
```

Where "0" is the desired vertical position for this bar.

Choose the vertical position you prefer.

To connect gantt bars:

1) Store the "AddGantt" or "AddGanttColor" function return longint:

```
Var tmp1, tmp2 : Longint;
tmp1:=GanttSeries1.AddGantt(EncodeDate(1997,1,1 ),
            EncodeDate( 1997,1,31 ),
              0,'Programming' );
tmp2:=GanttSeries1.AddGantt( EncodeDate( 1997,4,1 ),
             EncodeDate( 1997,4,30),
              0, 'Testing' );
```

2) Then use the NextTask property:

```
GanttSeries1.NextTask[ tmp1 ] := tmp2 ;
```

This will draw a line from 'Programming' gantt bar to 'Testing' bar. The "ConnectingLinePen" property is

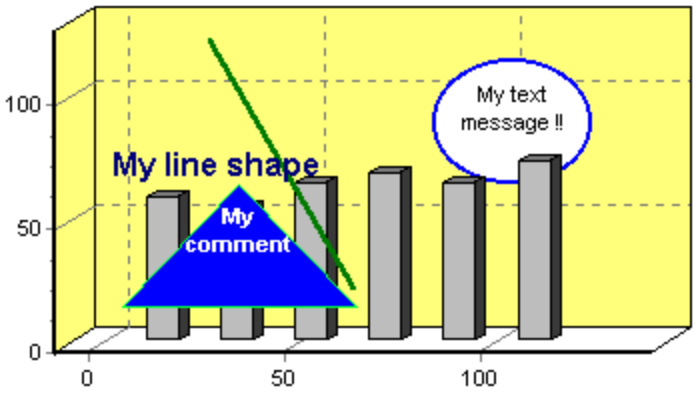
the pen used to draw lines.

Shape

See TChartShape for a full list of properties and methods

Shape Series are useful for defining or adding any additional information to your Chart, perhaps in runtime as a result of receipt of exceptional data. You may add text to any shape you add to your Chart and relate the shape to another Series.

Fig. 11.
Shape
Series



Each shape has two co-ordinates associated with it, top left and bottom right of the invisible rectangle that encloses the shape. You may add text to the box. These co-ordinates and messages could be updated at runtime by your code to dynamically put the shapes anywhere on your Chart.

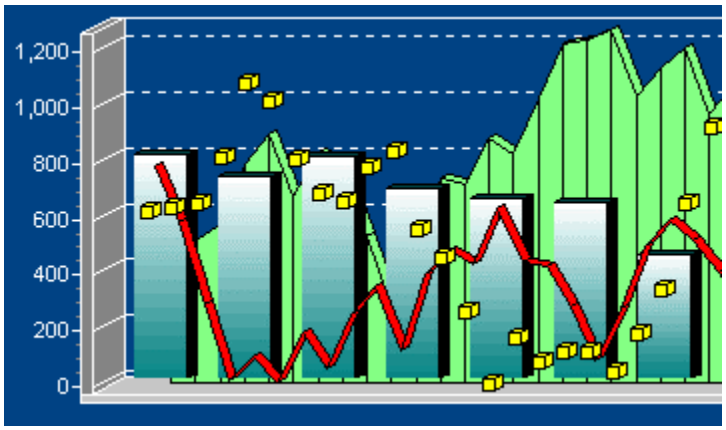
Combining Series

There is no practical limit to the number of Series that you can add concurrently to your Chart.

You may mix different Series types, almost any Series type with any other Series type. For certain combinations it is not practical as axis definition between Series may directly conflict. For those cases the Series not available are greyed out in the Series gallery so that you cannot mistakenly select them.

See the section on functions for more about combining Series types. Functions work with other Series to create and display algorithmic relationships.

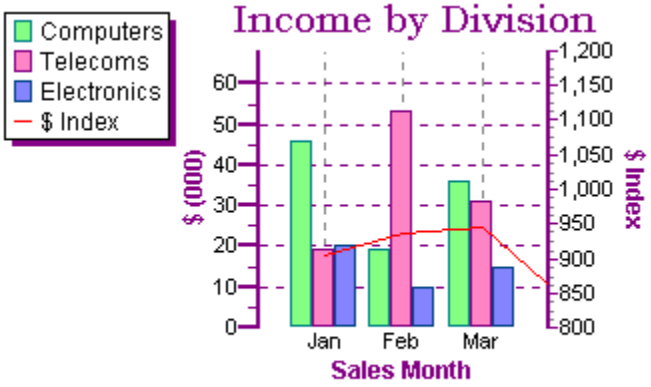
Fig. 12.
Combining Series



Example Series combination

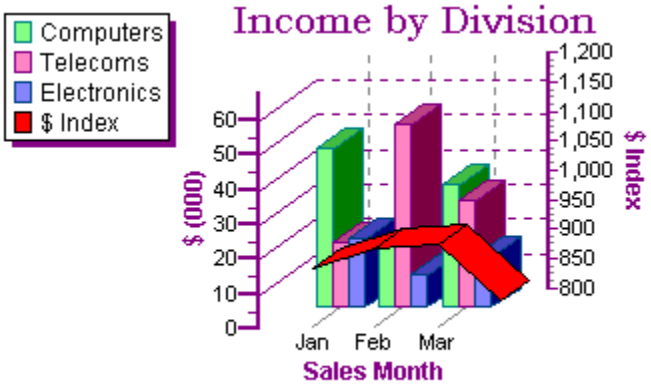
Combining Seriestypes in one Chart can add a great deal of information value. The following example shows the incomes by Division and puts the $ index in the same Chart to measure the effect of that external influence on incomes (perhaps no influence at all in this example !!).

Fig. 13.
2D Combining Series types



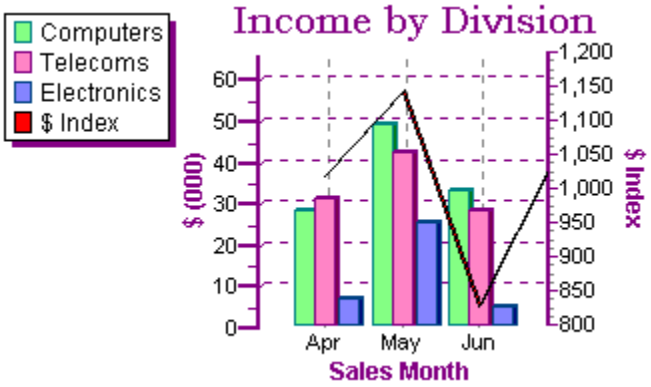
You may combine Series in 3D Charts. The previous example is represented in 3D below. The Chart looks attractive although a high degree of 3D perspective makes it more difficult to visualise the monthly $ index with Division incomes.

Fig.14.
3D combining Series types (60% 3D)

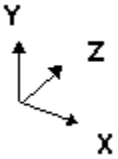


You could minimise the effect by reducing the extent of 'side' view (percentage 3D).

Fig. 15.
3D combining Series types (6% 3D)



Or you could modify code so that at runtime the Chart will put all Series in the same Z-plane. 3D Charts have a property called Z-Order (see figure) which controls the depth position of each Series.



It is possible to put all Series on the same plane although we advise caution as the effect may be confusing depending on which Series types you are combining.
The following line of code (name of Chart here **Chart1**) will put all Series in the same Z plane:

```
Chart1.ApplyZOrder:=False;
```

Adding a Chart to QuickReport

TeeChart integrates seamlessly with Qusoft's QuickReport. To add a TeeChart to a QuickReport follow these steps:

Drop a QuickReport on a form.

Using the QRChart icon on the QuickReport palette

Put a TQRChart onto the QuickReport worksheet.
You may now double click on the TQRChart to bring up the Chart editor. From this point forward you may work with the Chart as with any other TDBChart.
At runtime, access properties and methods for the Chart via the **TQRChart.Chart** property.

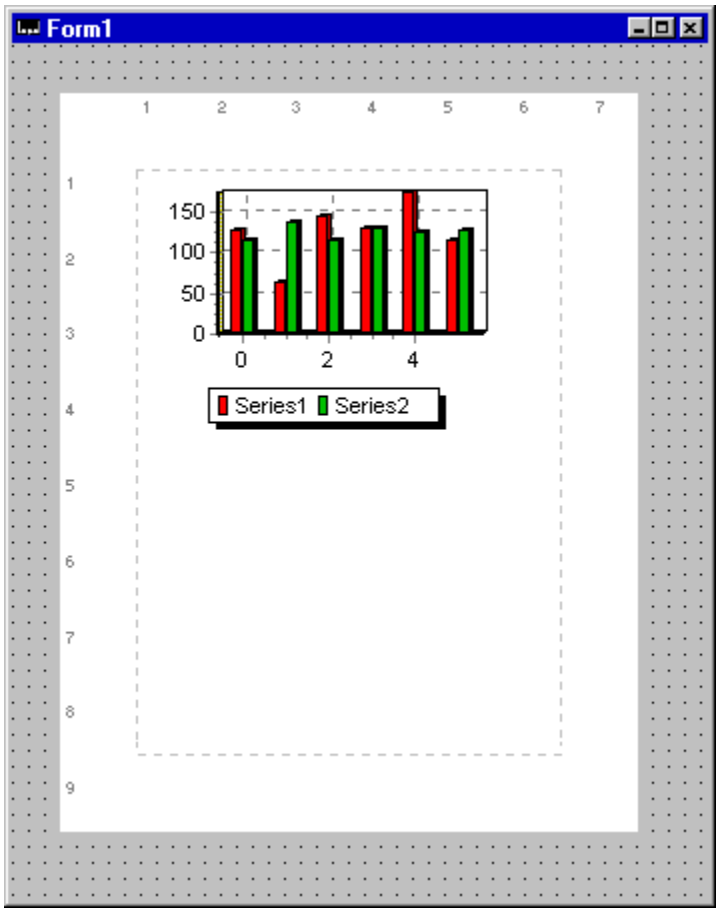


Fig. 1
TQRChart on a QReport at design time

Notes on working with QuickReport

Metafile

TeeChart uses Windows Metafile format for superior quality when outputting to a printer and for drawing onscreen. Also the quality of the printout is better if the Chart shape is maintained as nearly as possible to the default Chart rectangle form.

Clipping

TQRCharts do not support Series clipping. The may result in slight overlay of the Series onto the Chart axis if data points are adjacent to the axis.

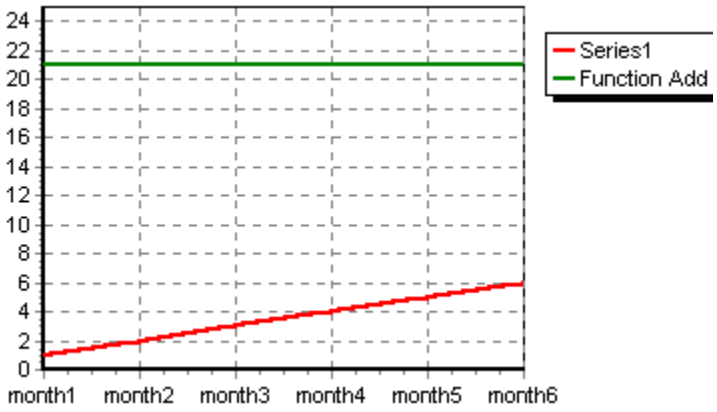
Standard functions

Add

See TAddTeeFunction for a full list of properties and methods

The Add function adds data from one or more Series. If we create a line Series ‘Series1’, create a line Series ‘Function Add’ and define Series ‘Function Add’ as Add of Series1 and do nothing more we will obtain a Chart with Series1 displayed and ‘Function Add’ as one flat line which is the sum of all values of Series1. In the figure the total of 1 + 2 + 3 + 4 + 5 + 6 = 21.

Fig. 1.
2D Add function with 1 Series input



We can modify the Series ‘Function Add’ to represent Add of Series1 by 2 point groupings (1+2), (3+4), (5+6). We use the period property (Chart Editor in the Data source page). Coded it looks like this:

```
Series2.FunctionType.Period:=2;
{where Series2 is the functionSeries}
```

Alternatively:

```
TeeFunction1.Period:=2;
{The name of the function is automatically allocated when the function is
  defined in the Chart editor - You will find it in the Object inspector}
```

Defining the period as 2, sets the Add function to add every 2 points. The period property adds enormous value to function Series.

Fig. 2.
Add function with period = 2

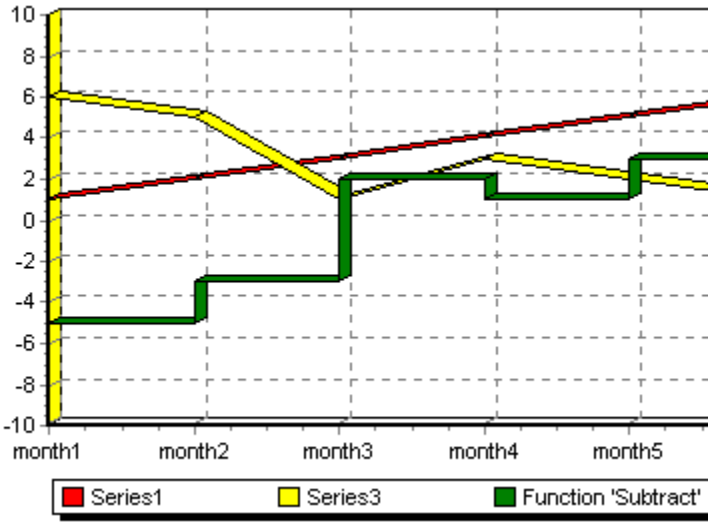
Subtract

See TSubtractTeeFunction for a full list of properties and methods

Subtract requires 2 input Series. With more than one Series in the function the default period sets to 1 axis point. the 2nd Series will be subtracted from the 1st Series in the list.

The following Chart is defined with ApplyZOrder:=False which forces all Series to draw in the same 3-Dimensional plane - The resulting Series overlay (a sort of optical illusion) in the Chart depends on the Series paint order.

Fig. 3.
Subtract function

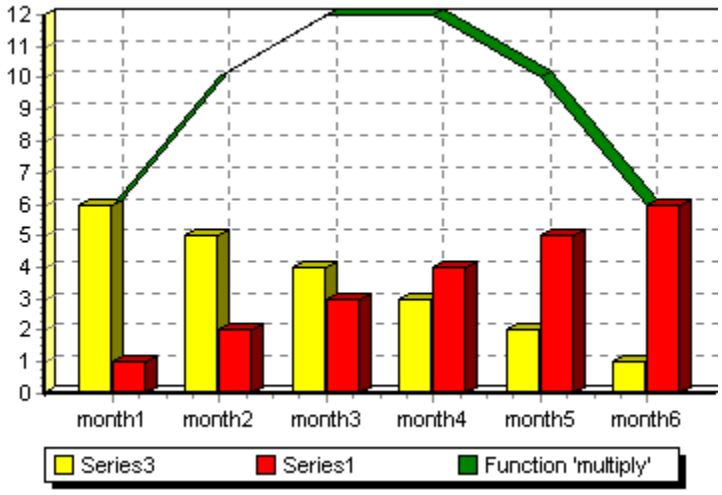


Multiply

See TMultiplyTeeFunction for a full list of properties and methods

The default period for the function ‘multiply’ is 1. You may add as many Series as you wish to the multiply function.

Fig. 4.
Multiply function



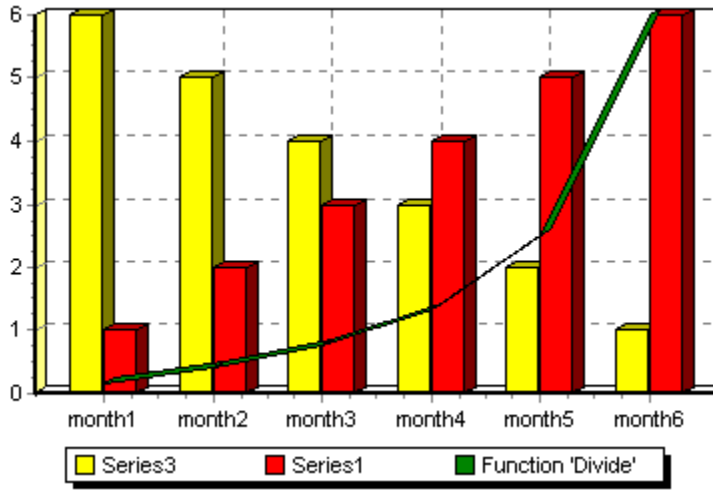
Divide

See TDivideTeeFunction for a full list of properties and methods

As divide requires at least 2 input Series the default period for the divide function is 1. The 2nd Series in the list is the denominator.

If you add more than 2 Series then the 1st will be divided by the 2nd then that is divided by the third, etc.…

Fig. 5.
Divide function

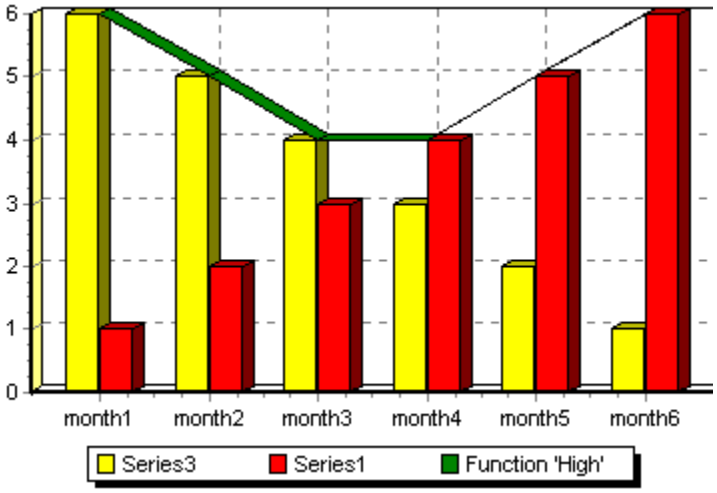


High

See THighTeeFunction for a full list of properties and methods

High accepts multiple input Series and will always display the highest point between those Series at each period point. (1 Series default period 0, 2 or more Series default period 1).

Fig. 6.
High function

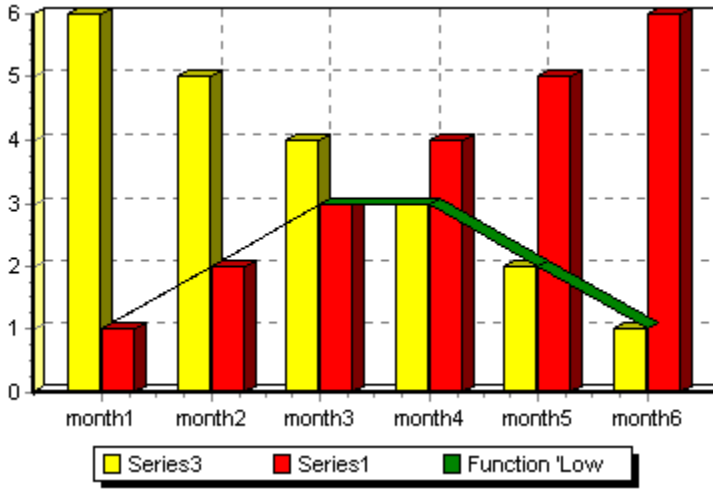


Low

See TLowTeeFunction for a full list of properties and methods

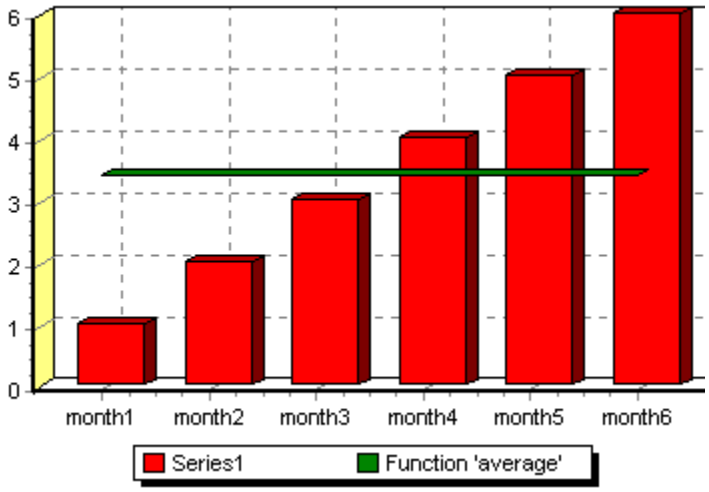
Low accepts multiple input Series. It will always display the lowest point between those Series at each period point. (1 Series default period 0, 2 or more Series default period 1).

Fig. 7.
Low function



Average
See TAverageTeeFunction for a full list of properties and methods

Fig. 8.
Average function



The default period for average with one Series is 0 (all) which will give you the average for that Series across the whole Chart. If you have more than one Series the period will be 1 axis point.

You may change the period to alter the frequency of the average curve.

Overview - Working with Charts and Series

This section describes some design considerations and conventions for working with TeeChart. The section **Working with Charts and Series** brings to your attention some points of interest that may help with the design of your application.

Topics in this section include:

Click events

Custom drawing on the Chart

Series manipulation

Printing Charts

Chart Zoom and Scroll

Real-time Charting and speed

Functions

To understand the design paradigm of TeeChart we need to separate conceptually the contents of the Chart -the data Series- from the Chart itself, e.g. its Axis format, Legend and Titles. We can define and work with Series across Chart boundaries, referencing Series components independently of the Chart. We may, however, copy and paste a Chart and it will copy with all its defined contents, Axis, Legends and Series

Two Chart components

There are 2 Chart components - The distinction serves simply to help cut down memory use for Charts that do not need to access a database.

Projects compiled with using only TChart components, **don't need** the Borland Database Engine DLL's (BDE). A new TChart will not contain configuration parameters for connecting to a database.

Delphi Version 4 and 5 offers the use of the TClientDataset as a Datasource. It does not use the BDE.

Chart subcomponents

The 2 Chart components have subcomponents. If, for example, you wish to access or modify the property of an axis of your Chart you will be modifying the property of one of the following subcomponents:

```
BottomAxis
TopAxis
LeftAxis
RightAxis
```

See TChartAxis to get a complete list of axis properties. Legend is another example of of a TChart subcomponent.

The Chart as backdrop to your Series

The Chart components provide the backdrop for your data Series. The overall ‘look’ of your Chart is controlled by the Chart properties accessible via the Chart Editor, the Delphi Object Inspector or by coding - see the online help for a complete list.

As a backdrop you have available to you the Delphi Canvas property and methods to calculate screen co-ordinates from values and vice-versa. The UDRAW.PAS example unit is a good reference point to see some examples of access to the canvas. In short, every TChart component has a "Canvas" property which is a standard Delphi TCanvas object.

Associating a Series with a Chart

Series associate with a Chart via a Series property called ParentChart. The Series are moveable between Charts by changing the Chart defined in the ParentChart property.

Via the Chart editor

After a Chart has been placed on a form, a right button click gives access to the Edit Chart menu (alternatively double-click). The first screen of the Edit Chart menu offers a list of button options to add, delete, clone, change type or title of a Series. Adding a data Series via this menu automatically associates the new data Series with the Chart.

Via the Object Inspector

Whilst browsing the list of components of a form in the Object Inspector you should see all Series added to any Chart on that form. Those Series components are not directly visible on the form but are nevertheless components with accessible properties.

If, for example, and you could try this with a simple test, you have 2 Charts on your form and have used the Chart editor to add a Series to one of the Charts - you could now access the property list of that Series in the Object Inspector and change the name of ParentChart to the other Chart on the form. On now opening the Chart editor for the new 'ParentChart' you will see the Series. Of course if you make this move for a Series defined with DataSource from a TDBChart to a TChart you will lose the ability to access the DataSource as the TChart is not database aware.

Via code

Use the ParentChart property to associate (or disassociate) a Series with a Chart.

Example:

In Form1, we'll create and show another Form (Form2), and assign Form1.LineSeries1 to Form2.ChartInForm2 :

```
With TForm2.Create(Self) do
try
  Self.LineSeries1.ParentChart := ChartInForm2 ;
  ShowModal ;
finally
  Free ;
end ;
```

That will show Form2 (containing a Chart component) and drawing Form1.LineSeries1.

Often used parameters

All the parameters associated with Charts are useful at the time you need them. We cover a few frequently used properties in detail below as an introduction. For a more detailed study of these and many other configurable parameters please refer to the remaing Topics of this section, **Working with Charts and Series**.

Label increment, % separation and angle

It may be difficult to fit all your axis labels onto the axis that displays onscreen - As an example, this is often true of date labels which are long and can have a high incidence of change on the axis.

Label Increments

TeeChart offers **Label increment** as a means to controlling the frequency of labelling on your Chart axis. You can access the property via the Chart Editor under **Axis** (select your axis) - **Scales** - **Desired increment**. The menu shows you a list of the standard options for time increments (e.g. One second, one minute … one day, etc.) Selecting 'one day' will cut out repetitions of date on the axis, showing the date label only once for the point range of that date.

Via code you may access the property as a property of the subcomponent for the relevant axis.

Example:

```
Chart1.BottomAxis.Increment := DateTimeStep[ dtOneHour ] ;
Chart1.RightAxis.Increment := 1000 ;
```

% Separation

If the labels on the axis have a look of rolling into one another you can increase the % separation which will forcibly increase the gap between them. In some cases this may, as a result, hide some labels. It will very much depend on your labels and whether your Chart users will have the option to 'resize' their

Chart.

The option is available in the Chart Editor under **Axis** (select your axis) - **Labels** - **% Separation.**

Setting Labels Separation to 0 % means no overlapping checking will be performed, thus all axis labels will be displayed.

Angle

If it is neither practical or possible to place the axis labels side to side along the axis another option is change the angle of the labels to 90º to the axis. This option is available in the Chart Editor under **Axis** (select your axis) - **Labels** - **% Angle**.

BitMaps (BMP) or Metafiles

TeeChart offers both the bitmap (BMP) and the Metafile format to save Charts. Two type of metafile formats are supported WMF (Windows Metafile Format) and EMF (Extended Metafile Format).

Bitmap format is used internally by TeeChart. It is quicker to draw onscreen than a Metafile. When requiring a Chart to be 'exported' to another application or to be embedded in a container application such as MSWord a Metafile format handles resizing better onscreen - TeeChart uses Metafile format with QuickReport.

Zoom

TeeChart offers as default zoom on all Charts. To obtain zoom at runtime hold the left mouse button and drag mouse toward down/right. You'll see a rectangle around the selected area. Release the left mouse button to Zoom. You can continue zooming again and again. To RESTORE (or UNDO) the zoom, drag a rectangle in the opposite direction (up/left).

Note

You may use mouseclicks to activate code associated with points in data Series. That OnClick behaviour will override zoom behaviour. You can set toggle between OnClick events and zoom with the CancelMouse property.

You may enable or disable zoom functionality in the Chart Editor - Chart page, General tab. The option, AllowZoom accesses the Chart property, also changeable by code:

```
Chart1.AllowZoom := False;
```

'False' deactivates zoom.

Zoom can also be obtained via coding. It is necessary to obtain the screen pixel co-ordinates for the area in which you wish to do the zoom.

Example 1:

Zoom an area with "pixel" co-ordinates:

```
Rect.Left := 123 ;
Rect.Top  := 67 ;
Rect.Right := 175 ;
Rect.Bottom:= 100 ;
Chart1.ZoomRect( Rect );
```

Example 2:

Zoom an area with point value co-ordinates:

You need first to translate from value to pixel co-ordinates. To do so, you can use the Axis or Series components.

```
Rect.Left := LineSeries1.CalcXPosValue(22.5);
Rect.Top  := LineSeries1.CalcYPosValue(5000);
Rect.Right := LineSeries1.CalcXPosValue(57.6);
Rect.Bottom:= LineSeries1.CalcYPosValue(15000);
Chart1.ZoomRect(Rect);
```

Backimage

You can add more information to your Chart or simply enhance its appearance by adding a bitmap as a

backdrop.

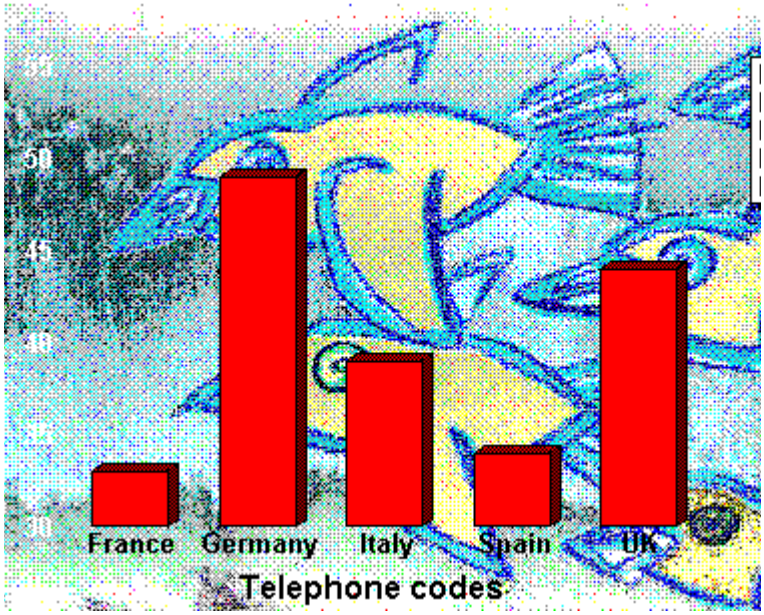


Fig. 1.
Bitmap as a Chart backdrop

You will find the parameters to define your Chart backdrop in the **General** tab of the Chart page of the Chart editor.

Click events

Topics in this section include:

OnClickSeries

OnClick

Series OnClick and OnDblClick

Chart OnClickSeries

The TChart event, OnClickSeries, permits access to any active Series in your Chart. Put the following code into your project to get an idea of the possibilities. To access the event and add the procedure definition to your project, select your Chart and open the Object Inspector. Select the events tab to find OnClickSeries. Double-click on OnClickSeries.

```
procedure TForm1.DBChart1ClickSeries(Sender: TCustomChart;
 Series: TChartSeries; ValueIndex: Longint; Button: TMouseButton;
 Shift: TShiftState; X, Y: Integer);
begin
 ShowMessage(' Clicked Series: '+Series.Name+' at point: '+
  inttostr(valueindex));
end;
```

The valueindex refers to the index of the Series data point (point, bar, etc.) within the Chart. You may use it to access the X and Y value. For example:

```
begin
    ShowMessage(' Clicked Series: '+Series.Name+' at point: '+
  Floattostr(Series.XValue[valueindex]) + ',' +
  Floattostr(Series.YValue[valueindex]));
end;
```

Chart OnClick

You may use the OnClick event of the Chart to get the same information.

```
procedure TForm1.DBChart1Click(Sender: TObject);
var t,tmp:Longint;
  x,y:Double;
begin
 Series1.GetCursorValues(x,y);
 for t:=0 to DBChart1.SeriesCount-1 do
 begin
  tmp:=DBChart1.Series[t].GetCursorValueIndex;
  if tmp<>-1 then
    ShowMessage(' Clicked Series: '+DBChart1.Series[t].Name+' at point:
  '+inttostr(tmp));
 end;
end;
```

Series OnClick and OnDblClick

The Series OnClick event catches click events at Series level. Thus for multiple Series Charts you are able to restrict access to the data of a specific Series. To access the OnClick event of the Series you must select the Series in the Object Inspector and go to the events tab.

```
procedure TForm1.Series1Click(Sender: TChartSeries; ValueIndex: Longint;
 Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
 ShowMessage(' hello: '+Sender.Name+' at point: '+inttostr(valueindex));
end;
```

Custom drawing on the Chart

If you require to add textboxes or other shapes relative to Chart axis the Shape Series may be the best choice. If the shape Series doesn't match your specific requirement you may draw your own lines and/or shapes on the Chart.

TeeChart offers access to the Chart area by axis or by screen pixel. The TeeChart demos, **UDraw.pas** and **Uhighlo.pas**, are useful references for custom drawing on a Chart.

Topics in this section include:

Calculating Co-ordinates

Axis' Value to Screen co-ordinate Methods

- CalcPosValue
- CalcPosPoint
- CalcSizeValue
- CalcYPos and CalcXPos

Series' Value to Screen co-ordinate Methods

- CalcPosValue
- CalcXPos and CalcYPos
- XScreenToValue and YScreenToValue

Chart Canvas

Writing to the Canvas

Internal bitmap

Repainting

When to draw ?

Chart Regions

Drawing

Calculating Co-ordinates

This chapter explains how to convert from Point co-ordinates to pixels and vice-versa. Also how to determine the exact co-ordinates for each graphics element in Chart components.

Point Values are expressed in user custom scales. Chart Axis are responsible for converting point values into suitable X and Y screen pixel co-ordinates for displaying them.

Both Axis and Series components have several functions to convert from screen co-ordinates (in pixels) to Axis or points values (in user-defined units).

The difference between using Axis or Series conversion functions is that Axis will only interpret co-ordinates for the topmost position in 3D mode, while Series will adjust co-ordinates to their Z order position.

Note: Using conversion functions is only valid after a Chart component has been drawn, either to screen or to a private hidden Canvas.

Axis' Value to Screen co-ordinate Methods

CalcPosValue

You can calculate the screen position for a given value using any Axis:

```
Var MyPos : Longint ;
MyPos := Chart1.LeftAxis.CalcPosValue( 100.0 );
```

MyPos holds now the pixel co-ordinate for “100.0” on Chart1 Left Axis (Vertical), being “100.0” a floating value in Axis scales.

You can use this co-ordinate to custom draw or to calculate mouse clicks. See chapters below.

If you want to convert from pixel co-ordinates to Axis values, you can use the opposite function:

CalcPosPoint

```
   Var MyValue : Double ;
   MyValue := Chart1.LeftAxis.CalcPosPoint( 100 );
```

MyValue holds now the Axis value co-ordinate for “100” on Chart1. Left Axis (Vertical), being “100” a screen pixel co-ordinate.

Note: Pixel co-ordinates start from 0, being 0,0 the Chart origin point. This is valid for screen, but when drawing to metafiles, drawing to custom canvases or printing, Chart origin point can optionally be different than 0,0.

The Chart1.ChartBounds property returns the origin and ending co-ordinates for the Chart bounding rectangle.

CalcSizeValue

Axis have another function to calculate how much screen space represents a given Axis range:

```
  Var Space : Longint ;
  Space := Chart1.LeftAxis.CalcSizeValue( 1000 );
```

Remember Axis can be DateTime and you can for example convert a Date Range period in pixels:

```
  Var Space : Longint ;
  Space := Chart1.BottomAxis.CalcSizeValue(
  EncodeDate( 1997,12,31) - EncodeDate( 1997,1,1) );
```

CalcYPos and CalcXPos

You could use CalcYPos and CalcXPos. When drawing using the XPos and YPos co-ordinates remember that the co-ordinate 0,0 is Top,Left of the Chart rectangle, ChartRect. ChartRect is the area enclosed by the 4 axis of the Chart.

The following example draws a line from an arbitary point from the Y-axis across the Chart. Note the use of canvas properties.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyHalfwayPoint, YPosition:longint;
Begin
With Series1.YValues do
  MyHalfwayPoint:=round(((MaxValue-MinValue)* 0.5) + MinValue);
{ then calculate the Screen Pixel co-ordinate of the above value }
    YPosition:=DBChart1.LeftAxis.CalcYPosValue(MyHalfwayPoint);
  With DBChart1.Canvas do
   begin
    { change pen and draw the line avoiding the
    3D areas and axis of the Chart - Height3D,Width3D}
    Pen.Width:=3;
    Pen.Style:=psSolid;
    Pen.Color:=clBlack;
     with DBChart1 do
     begin
    MoveTo(ChartRect.Left,YPosition);
    LineTo(ChartRect.Left+Width3D,YPosition-Height3D);
    LineTo(ChartRect.Right+Width3D,YPosition-Height3D);
     end;
    end;
end;
```

Series' Value to Screen co-ordinate Methods

Series have similar methods for converting co-ordinates to point values and vice-versa. The main difference is that by using the Series method you don't need to know the exact Axis component for calculations.

This is a big advantage when having Series associated to Right or Top Axis, or multiple Series associated to each Axis.

CalcPosValue

This code calculates where in the screen the Series1 point with value 1000 is located:

```
  Var MyPos : Longint ;
  MyPos := Series1.CalcPosValue( 1000 );
```

or...

```
  MyYPos := Series1.CalcPosValue( Series1.YValue[ 0 ]
       ) ; { <-- first point }
```

CalcXPos and CalcYPos

You can calculate both X and Y co-ordinates for a specific point or for a specific point value:

```
  MyXPos := Series1.CalcXPos( EncodeDate( 1997, 12, 31) ) ;
```

or...

```
  MyXPos := Series1.CalcXPos( Series1.XValues.Last );
    { <-- last point }
```

XScreenToValue and YScreenToValue

To convert from screen pixels to point values, use:

```
  Var MyValue : Double ;
  MyValue := Series1.YScreenToValue( Y ) ;
```

(and XScreenToValue for horizontal co-ordinates).

You can query the point under a given pair of XY screen co-ordinates using the Series.Clicked function. (See Mouse Clicks chapter).

Chart Canvas

Chart.Canvas is a standard Delphi Canvas. You may control the appearance of your Chart using Canvas properties. See TCanvas help in Delphi for a full list of properties.

Writing to the Canvas

The following example divides the area of the backdrop of the Chart rectangle into 5 equal segments and colours them according to the colour array.

```
procedure TDrawForm.LineSeries1BeforeDrawValues(Sender: TObject);
Const
  MyColors:array[1..5] of TColor=
  ( clNavy,
   clGreen,
   clYellow,
   clRed,
   $00000080 { very red }
   );
var t,partial:Longint;
  tmpRect:TRect;
  With Chart1 do
  Begin
    { we will divide the total Chart width by 5 }
    tmpRect:=ChartRect;
    tmpRect.Right:=tmpRect.Left;
    partial:=ChartWidth div 5;
```

```
    { change the brush style }
    Canvas.Brush.Style:=bsDiagCross;
    Canvas.Pen.Style:=psClear;
    { for each section,fill with a specific color}
    for t:=1 to 5 do
    Begin
     { adjust the rectangle dimension }
      tmpRect.Right:=tmpRect.Right+partial+1 ;
      { set the brush color }
      Canvas.Brush.Color:=MyColors[t];
      { paint !!! }
      With tmpRect do
      Canvas.Rectangle( Left+Width3D,Top-
    Height3D,Right+Width3D,Bottom-Height3D );
         { adjust rectangle }
         tmpRect.Left:=tmpRect.Right;
      end;
    end;
end;
```

Internal bitmap

TChart components have an internal bitmap object, which is used when drawing as a hidden “buffer”. When drawing is finished, this “buffer” is copied to the screen video memory to display it.

TChart Canvas property returns the internal bitmap Canvas object.

Note: BufferedDisplay property controls if the internal bitmap is used. Using this bitmap no flicker occurs when redrawing real-time Charts. Also, some Charts with many points can take advantage of faster drawing speed on memory bitmaps instead of direct drawing to slower video card memory chips.

Drawing to a bitmap Canvas is exactly the same as drawing to another “real” Canvas in terms of source code transparency and results.

Note: When drawing to a metafile or printing, the Chart Canvas property refers to metafile or printer Canvas objects. No bitmap is used in these cases.

After a Chart has been drawn onto the internal bitmap, and its copied onto the screen canvas, Chart Canvas property refers to the original “real” Chart Canvas.

Repainting

You should call Chart1.Repaint or Series1.Repaint to force a Chart component to draw again.

When to draw ?

The order in which you draw to the Canvas is important.

Best place to custom draw over a Chart component is at

If you wish Custom drawn items to appear above Chart Series you should use the Chart **OnAfterDraw** event. The Chart1.OnAfterDraw event is fired each time the Chart component is redrawn, just before copying the internal bitmap to screen.

You can place Custom drawn items above the Chart grid and below the Chart Series by placing your code in the Series OnBeforeDrawValues event. (See UDRAW.PAS example unit).

Advanced: For more control you might want to create your own Chart class and override the several Draw... virtual methods.

Chart Regions

The biggest rectangle around a Chart is at ChartBounds property. ChartWidth, ChartHeight, ChartXCenter and ChartYCenter are pre-calculated co-ordinates based on ChartBounds property.

Axis are drawn inside this space. The ChartRect property returns the bounding rectangle for Axis (same in 2D and 3D).

Chart Legend has the RectLegend public property which defines the Legend rectangle bounds.

Chart Title and Foot have TitleRect public properties.

In 3D Charts, each Series is assigned a specific section among the “Z” (depth) axis.

The Chart Width3D and Height3D are the dimensions of the

3D depth, in pixels. SeriesWidth3D and SeriesHeight3D are the dimensions for each individual Series in a Chart.

Drawing

Lets start drawing with a basic example.

This code draws an horizontal line just at the middle of Chart1:

```
procedure TForm1.Chart1AfterDraw(Sender: TObject);
begin
 With Chart1 do
    begin
         Canvas.Pen.Color:=clYellow;
         Canvas.MoveTo( ChartBounds.Left,
         ChartYCenter );
         Canvas.LineTo( ChartBounds.Right,
    ChartYCenter );
    end;
end;
```

Drawing inside Axis space:

```
procedure TForm1.Chart1AfterDraw(Sender: TObject);
begin
 With Chart1 do
   begin
    Canvas.Pen.Color:=clYellow;
    Canvas.MoveTo( ChartRect.Left, ChartYCenter );
    Canvas.LineTo( ChartRect.Right, ChartYCenter );
   end;
  end;
```

Drawing a line at each point in Series1 :

```
procedure TForm1.Series1AfterDrawValues(Sender: TObject);
var t ,x,y : Longint ;
begin
    for t := 0 to Series1.Count - 1 do
    begin
       x:=Series1.CalcXPos( t );
    y:=Series1.CalcYPos( t );
    Chart1.Canvas.MoveTo( x-8, y-8 );
    Chart1.Canvas.LineTo( x+8, y+8 );
   end;
end;
```

Note: Drawing Text

Always set Chart1.Canvas.Font.Height to a negative value

instead of using Font.Size if want same font sizes on screen

and on printer or metafiles.

See UHIGHLO.PAS and UDRAW.PAS units for example of custom drawing text.

Working with Axes

Topics in this section:

Setting Axis scales

Chart components have five axes: LeftAxis, RightAxis, TopAxis, BottomAxis and DepthAxis. Each axis is an instance of TChartAxis component class.

Axes are responsible of calculating pixel co-ordinates for Series points and to allow any valid range so scroll and zoom can be always performed. As new Series are inserted, or new points are added to the Series, Axes recalculate, by default, their Minimum and Maximum values.

You can turn off automatic recalculation of Axis scales by setting the Automatic property to false:

```
Chart1.LeftAxis.Automatic := False ;
```

Also, both the Axis Minimum and Axis Maximum values can optionally be independently automatic or not.

```
Chart1.LeftAxis.AutomaticMaximum := False ;
Chart1.LeftAxis.AutomaticMinimum := True ;
```

You can change Axis scales using the Minimum and Maximum properties:

```
  With Chart1.LeftAxis do
  begin
    Automatic := False ;
    Minimum:=     0 ;
    Maximum:= 10000 ;
  end;
```

or using the Axis SetMinMax method:

```
Chart1.LeftAxis.SetMinMax( 0, 10000 );
```

DateTime Axis

Note:

An Axis contains DateTime scales when the associated Series components have XValues.DateTime or YValues.DateTime properties True. There is no DateTime property for Axis.

Changing scales in DateTime Axis is the same as for non-datetime values:

```
With Chart1.LeftAxis do
  begin
    Automatic := False ;
    Minimum:= EncodeDate( 1990, 3,16 ) ;
    Maximum:= EncodeDate( 1996, 5, 24 );
  end;
```

Logarithmic Axis

An Axis can be set to Logarithmic only if Axis Minimum and Maximum are greater than or equal to zero. This is the only difference between setting linear and logarithmic scales.

Note:

Axis Labels are not displayed in logarithmic increments. You can generate custom Axis Labels (see chapter below).

Inverted Axis

An Axis can be Inverted so Axis Minimum and Maximum are swapped. We suggest you use Inverted:=True as little as possible as it can give misleading results due to its (normally) rarity of use.

Axis Styles & Increment

Axis can be displayed in several ways, with tick lines or without,

with grids or without, with Labels or without, and you can customise all formatting properties such as colours, fonts and pen styles. The Axis Increment property controls the number of grid lines and labels and the distance between them.

By default it's zero, meaning Axis will automatically calculate the Labels increment. Setting the Axis Increment property is independent of setting Axis scales. You can have automatic Axis Maximum and Minimum and a fixed Increment.

The Increment property grows automatically until all Axis Labels can be displayed. If you don't want this automatic feature, set the Axis LabelsSeparation to zero:

```
Chart1.LeftAxis.LabelsSeparation := 0 ;
```

Warning:

When LabelsSeparation is zero, no checking is performed against

labels size, so you must take care labels will not overlap each other.

The following code sets the vertical Axis Increment to 30:

```
Series1.Clear;
Series1.AddArray( [ 20, 50, 120 ] );
Chart1.LeftAxis.Increment:= 30;
```

By default, the first Axis label starts at nearest Increment. Setting RoundFirstLabel to False makes labels to start at Axis Maximum:

```
Chart1.LeftAxis.RoundFirstLabel := False ;
```

DateTime Increment

Use the predefined DateTimeStep array of constants to determine the Axis Increment on DateTime axis:

```
Chart1.BottomAxis.Increment := DateTimeStep[ dtOneMonth ] ;
```

And set the ExactDateTime property to True if you want Axis Labels to be at exact datetime boundaries, for example at 1st day of month.

```
Chart1.BottomAxis.ExactDateTime := True;
```

Note:

Logarithmic axis use the Increment property as linear.

Grid lines

Axis Grid lines are displayed at each Increment position, or at each Axis Label position. The Axis TickOnLabelsOnly property controls this feature:

```
Chart1.BottomAxis.TickOnLabelsOnly := False ;
```

Axis Labels

There are several Axis labelling styles. The Axis LabelStyle property control axis labelling modes:

```
Chart1.BottomAxis.LabelStyle := talValue ;
```

Possible values are:

talValue	Labels display Axis scales
talMark	Labels display Series Point Marks
talText	Labels display Series XLabels
talNone	No labels displayed
talAuto	Style is automatically calculated

When LabelStyle is talText, TeeChart will set it automatically to talValue if Series have no XLabels.

For talMark and talText styles, axis labels are displayed exactly at Series point positions, thus not using the axis Increment property.

You can customise labels text by using the OnGetAxisLabel event:

```
procedure TForm1.Chart1GetAxisLabel(Sender: TChartAxis;
 Series: TChartSeries; ValueIndex: Integer; var LabelText: String);
begin
   if Sender=Chart1.LeftAxis then
     if ValueIndex < 2 then LabelText :=” ;
end;
```

Advanced: The UAXISLAB.PAS example unit shows how to customise both Axis Labels text and position, using the OnGetNextAxisLabel event.

CustomDraw (Axis)

CustomDraw adds more axes to your Chart - As many as you wish. CustomDraw adds an axis as a ‘copy’ of an existing axis at a new location.

The MULAXIS.PAS example unit draws two custom Axis inside a Chart. Available Axis methods for custom drawing are: CustomDraw and CustomDrawMinMax.

The following example populates a line Series with random data. And creates 2 additional axis. There are 3 configurable locations PosLabels, PosTitle and PosAxis to place, respectively, the labels, title and the axis. The last parameter, GridVisible , which is boolean, defines whether to extend the axis grid to the new axis.

```
procedure TForm1.FormCreate(Sender: TObject);
var
  t:integer;
begin
   Series1.XValues.DateTime := False;
   Chart1.BottomAxis.Increment:=0;
   For t:= -10 to 10 do
     Series1.addXY(t,Random(100),'', clTeeColor);
end;
procedure TForm1.Series1AfterDrawValues(Sender: TObject);
var posaxis,percent:longint;
begin
 percent:=50;
 With DBChart1 do
 begin

  PosAxis:=ChartRect.Left+Trunc(ChartWidth*Percent/100.0);
  LeftAxis.CustomDraw(posaxis-10,posaxis-40,posaxis,True);
    PosAxis:=ChartRect.Top+Trunc(ChartHeight*Percent/100.0);
    BottomAxis.CustomDraw(posaxis+10,posaxis+40,posaxis,True);
 end;
end;
```

Series manipulation

This chapter documents how to manipulate Series and how to manipulate Series points and other Series internal data.

Note: Advanced. Series and values code is mainly located at TEENGINE.PAS unit.

Topics in this section:

Creating Series at runtime.

Series can be created at runtime just as like any other Delphi component:

First you need a Series variable:

```
Var MySeries : TBarSeries ;
```

{Then create the component:}

```
MySeries := TBarSeries.Create( Self );
```

{And then assign it to the desired Chart component:}

```
MySeries.ParentChart := Chart1 ;
```

{Now you can add points to MySeries or do whatever you can do with a design-time created Series.}

Shortcut:

{If you don't need access to MySeries, the above code can be reduced to just one line of code:}

```
Chart1.AddSeries( TBarSeries.Create( Self ) );
```

{Class references:

If you don't know the Series type, you can use a Series class reference:

First declare the Series class variable:}

```
Var MyClass : TChartSeriesClass;
```

{Then set the desired Series class type:}

```
MyClass := TBarSeries ;
```
{Now you can create the Series component:}
```
Chart1.AddSeries( MyClass.Create( Self ) );
```

Series array property

Chart components store all Series in SeriesList property, a Delphi TList object.

You have read-only access to this list in three ways:

A) Using the SeriesList property:
```
  MySeries := Chart1.SeriesList [ 0 ]
```
B) Using the Series array property:
```
  MySeries := Chart1.Series [ 0 ]
```
C) Or using the Chart1 default property:
```
  MySeries := Chart1 [ 0 ]
```
Either way does the same.

SeriesCount property

The Chart1.SeriesCount property returns the number of Series in SeriesList.

You can use SeriesCount to traverse all Chart Series:
```
for t := 0 to Chart1.SeriesCount - 1 do
With Chart1.Series [ t ] do
begin
  SeriesColor := clBlue ;
end;
```

Deleting Series

Series can be hidden in three ways:

A) Setting the Series Active property:
```
  Series1.Active := False ;
```
B) Removing the Series from their parent Chart:
```
  Series1.ParentChart := nil;
```
C) Destroying the Series completely:
```
  Series1.Free ;
```

Changing the Series Z order at runtime.

In 3D mode (when Chart1.View3D is True), all Series are assigned a Z order position. That is, the order where Series will be drawn, starting with the far most Series on the Chart 3D space.

You can control the order Series will be drawn, using these methods:
```
Chart1.ExchangeSeries( 0, 1 );
```
or...
```
Chart1.SeriesUp( Series1 );
Chart1.SeriesDown( Series1 );
```
The Series.ZOrder integer property returns the corresponding Z position for the Series.

Some Series in certain modes share the same ZOrder, like Stacked BarSeries. You can check if more than one Series has the same Z order using these functions:
```
if Series1.FirstInZOrder then .....
if Series1.MoreSameZOrder then .....
```

Adding Points

Every Series type has, at least, 2 values for each point. These values are designed as X and Y point co-ordinates.

Note: Values are defined as “Double” floating point variables.

Extended Series types have more than 2 values, like BubbleSeries has X, Y and Radius values for each point.

So, each Series type has its appropriate method to add points, although the most common Series types like Line, Bar, Point and Area share the generic AddY method to add points.

The following code empties a TPieSeries and adds some sample values to it:

```
Series1.Clear ;
Series1.Add( 1234, 'USA', clBlue );
Series1.Add( 2001, 'Europe', clRed );
```

For extended Series types, please refer to each specific Series reference to know which method should be used to add points.

The Series AddArray method can be used in some situations:

```
Series3.Clear;
Series3.AddArray([ 1234, 2001, 4567.12 ] );
```

AddArray does not Clear the Series before adding points.

You can express points as constants or variables:

```
Series3.AddArray([ Table1Sales.AsFloat, 123,  MyTotal ] );
```

Null Values

In some circumstances you might have no values for specific points. You should then add these points as “zero” or add them as “null” values.

Null values will not be displayed, while “zero” values will be displayed as usual.

The following code adds several points and a null point:

```
With Series1 do
begin
 Clear ;
 AddY( 10, 'John', clGreen );
 AddY( 20, 'Anne', clYellow );
 AddY( 15, 'Thomas', clRed );
 AddNull( 'Peter' );
 AddY( 25, 'Tony', clBlue );
 AddY( 20, 'Mike', clLime );
end;
```

Each Series type will handle null values in different ways.

Bar, HorizBar, Line, Area and Point do not display null points.

PieSeries use null values as “zero”.

Controlling Points Order

Points can be optionally sorted either by their X values or Y values. The Series.XValues and Series.YValues Order property controls the points Order:

```
Series1.XValues.Order := loAscending ;
```

Possible values are: loNone, loAscending or loDescending.

By default, XValues Order is set to loAscending, and YValues Order to loNone, which means new added points are ordered by their X co-ordinate. For non XY Charts, the X co-ordinate is always the point position, starting from zero.

The point Order is used by Series components to draw their points.

Note: Order must be set before points are added to the Series.

You can change the Order property at run-time, followed by a call to Sort method:

Example:

Drop a TChart onto a Form, add a Line Series.

Drop a TButton and assign this code to Button1Click:

```
Series1.AddArray([ 5, 2, 3, 9]);
```

Now drop another TButton and add this code to Button2Click:

```
With Series1 do
begin
  YValues.Order:=loAscending;
  YValues.Sort;
  Repaint;
end;
```

Now execute and press Button1 to fill Series1, and press Button2 to see Series1 points drawn on Series1 YValues Ascending order, but having the original X co-ordinates.

Note: If you aren't using X co-ordinates, one more line of code is required.

Drop a new TButton (Button3) and add this code to Button3Click:

```
Series1.XValues.FillSequence;
Series1.Repaint;
```

Now points will be "re-numbered", starting from zero among Series1 XValues axis. This will re-order points so now they will be drawn as if they were originally added to the Series in their new order.

This "two-step" point sorting allows Line Series to draw with vertical orientation.

XY Points

Adding X co-ordinates to points makes Series components calculate user specific horizontal point positions.

Note: Bar Charts can be difficult to interpret with X co-ordinates.

Pie Series do not allow X co-ordinates.

To add X co-ordinates, simply use the AddXY method:

Drop a TChart, add a Point Series:

```
With Series1 do
begin
 Clear ;
 AddXY( 10, 10, 'Barcelona', clBlue );
 AddXY( 1, 10, 'San Francisco', clRed );
end;
```

Note: If you use a Bubble Series, use the TBubbleSeries AddBubble method.

Remember to set XValues.Order to loNone if don't want points to be sorted on their X co-ordinate.

Point Limits

16bit Delphi 1.0

Maximum 16380 Series per Chart, and 16380 points per Series.

32bit Delphi

Maximum 134217727 points per Series and same for Series per Chart.

Deleting Points

Simply call the Series.Delete method, passing the point index as the argument. Point index starts at zero.

```
Series1.Delete( 0 ); { <-- removes the first point in Series1 }
Series1.Delete( Series1.Count - 1 ) ; { <-- removes the last point in
  Series1 }
```

An exception “List out of bounds” will be raised when attempting to Delete a non-existing point, so always check there are enough points in the Series prior to delete them:

```
if Series1.Count > MyIndex then Series1.Delete( MyIndex );
```

Calling Delete forces re-calculation of Functions and repainting of the Chart.

Series Points

Retrieving and modifying Points

Once points have been added, you can retrieve their co-ordinates or change them.

The XValues and YValues array properties can be used:

```
Var MyValue : Double ;
  MyValue := Series1.YValues[ 0 ] ;  { <-- retrieves
             the first Y value }
```

You can traverse these arrays to perform calculations:

```
  Var MyTotal : Double ;
    t : Integer ;
  MyTotal := 0 ;
  for t:= 0 to Series1.Count - 1 do
    MyTotal := MyTotal + Series1.YValues[ t ] ;
  ShowMessage( FloatToStr( MyTotal ) );
```

Extended Series types have additional array properties, such as

BubbleSeries RadiusValues. You can access these properties in

the same way as with XValues or YValues arrays:

```
if BubbleSeries1.RadiusValues[ Index ] > 100 then .....
```

Modifying point values can be performed using the above properties:

```
  Series1.YValues[ 0 ] := Series1.YValues[ 0 ] + 1 ;
  Series1.RefreshSeries ;
```

Locating Points

The XValues and YValues Locate function searches a specific

value in the List and, if found, returns the value Index, starting

from zero.

```
  Var MyIndex : Integer ;
  MyIndex := Series1.YValues.Locate( 123 );
  if MyIndex = -1 then
   ShowMessage(‘ 123 not found in Series1 !! ‘ )
  else
   ShowMessage(‘ 123 is the ‘+IntToStr( MyIndex+1 )+’ th point in Series1 !!
  ‘ );
```

Point Statistics

XValues and YValues properties maintain the following statistical figures:

Total	The sum of all values in the list.
TotalABS	The sum of all values as absolute (positive).
MaxValue	The maximum value in the list.
MinValue	The minimum value in the list.

You can call manually to RecalcMinMax method to recalculate MinValue and MaxValue. Total and TotalABS are maintained automatically.

These values are used to speed up several Axis scaling calculations and are used as helpers for percent calculations.

TChartValueList object has several other methods and properties to manipulate point values. Please refer to online help documentation.

Notifications

Whenever points are added, deleted or modified, the Series generates internal notification events. These events are used to recalculate Series that depend on other Series points.

Advanced: You can use RecalcOptions property to control when recalculations are performed.

Point Colours

All Series maintain an internal List of colours. One for each point in the Series.

You can access this list with ValueColor array property, to retrieve or change point colours:

```
Var MyColor : TColor ;
MyColor := Series1.ValueColor[ 0 ] ;
Series1.ValueColor[ 1 ] := clBlue ;
```

TeeChart defines a generic colour constant named: clTeeColor.

Points with clTeeColor colour will be drawn using the SeriesColor colour.

Delphi predefines constants for basic colours (clBlue, clRed, etc.). Colours can also be expressed in RGB format. Using a video colour depth of 16k colours or more results in better colour matching.

Point Labels

Each point has an associated text, called XLabel, and declared as a Delphi string.

Point Labels are used in Axis Labels, Chart Legend and Point Marks.

Labels are stored at Series XLabel array property.

You can access and modify XLabel point texts:

```
Series1.XLabel [ 0 ] := 'Sales' ;
```

Changing the Series type at runtime - Advanced

Every Chart type corresponds to a different Delphi Component.

Changing a Series type involves changing the Series component class.

That means a new Series of the new class must be created, old Series properties should be assigned to the new instance, and finally the old Series must be destroyed.

All this happens automatically when you manually change a Series type at design-time using the Chart Editor Dialog and the Gallery.

You can change a Series type at run-time calling:

```
ChangeSeriesTypeGallery(Self, MySeries );
```

Warning:

You should change your private Series components.

You can change also Series components owned by the Form,

but only if not using them after.

```
Var MySeries : TChartSeries ;
MySeries := Series1 ;
ChangeSeriesTypeGallery(Self, MySeries );
```

Printing Charts

This chapter explains how Charts are printed and which properties and methods are used to control the printing process.

Note:

Advanced. Printing methods are located in the CHART.PAS unit.

Topics in this section:

Printing

Margins

Resolution

Print, PrintLandscape....

PrintPartial

Multiple Charts per Page

Windows and printers limitations

Clipping

Rounding circles

Rotating fonts

Dotted Pen styles and Pen Width

From FAQ - Practical issues when printing TeeCharts

Introduction

Design issues

How to Print Proportionally?

Other problems

Printing Reference

More information:

Printing

Margins

When printing, you can specify which margins on the paper page should be left blank. The public PrintMargins:TRect property stores the desired paper margins expressed as percents of total page dimensions.

```
PrintMargins:=Rect(15,15,15,15); { default 15 % printing margins }
```

The TChart.ChartPrintRect:TRect function returns, after applying the printing margins, the space where the Chart component will be drawn expressed as logical canvas units (pixels or dots).

With PrintMargins you can define any area of any size inside the page.

Note:

When changing paper orientation, margins are recalculated.

Resolution

Charts are printed in metafile (WMF in 16bit, EMF in 32bit) format.

Metafiles are scaleable vector formats, so a “wysiwyg” effect can be achieved if sending to the printer how a Chart looks on screen. However, you might want to exploit the printer’s bigger resolution capabilities versus screen displays.

The TChart.PrintResolution integer property controls how a Chart is scaled when sending it to the printer. By default PrintResolution is zero. Setting it to a negative value in the percentage range from zero to -100 makes the Chart proportionally bigger so there’s more space for Axis Labels. Smaller fonts are used as they will be clearer on paper than on screen.

Setting it to a positive value makes font sizes bigger.

Print, PrintLandscape....

Several methods exist to print a Chart component:

```
Print;
PrintRect(Const R:TRect);
PrintOrientation(AOrientation:TPrinterOrientation);
PrintPortrait;
PrintLandscape;
```

All the above methods will do the same, print a Chart on a new page and **eject** (form feed) the page:

```
Chart1.Print ;
```

Will print a Chart1 in the current printer’s paper orientation.

With PrintRect you can specify a custom Chart size and location. The default location and size is determined by ChartPrintRect function, which applies PrintMargins margins.

Delphi’s Printers unit contains the TPrinter object used for all Chart print methods.

PrintPartial

More advanced printing control can be obtained using the following methods:

```
Procedure PrintPartial(Const PrinterRect:TRect);
Procedure PrintPartialCanvas( PrintCanvas:TCanvas;
                 Const PrinterRect:TRect);
```

Both will print a Chart component, BUT WILL NOT create a new printer page or eject it. You can print both your own text and graphics and Chart components on the same paper page.

Multiple Charts per Page

An example of PrintPartial method is located at BASIC.PAS unit under TeeDemo project.

Four Chart components are printed on a single page, optionally with more resolution.

Windows and printer limitations

Metafiles are very good as they are small and fast and scaleable.

Some limitations occur when using metafiles, as described in Microsoft Knowledge Base (www.mskb.com). TeeChart inherits those limitations:

Clipping

Clipping is stored in physical co-ordinates in metafiles. This means moving or scaling a clipped metafile will not scale or move the clipping region, giving undesirable results.

TeeChart does not draw partial points, so no clipping. Drawing on zoomed Charts will probably show partial points outside the Chart axis.

Rounding circles

As metafiles can be scaled, circles will be converted to ellipses when scaling.

Rotating fonts

Rotated fonts can not be exactly aligned on non-proportionally scaled metafiles.

Dotted Pen styles and Pen Width

Non solid pens (dots, dashes), can be drawn as solid with scaled metafiles.

From FAQ - Practical issues when printing TeeCharts

This section explains how TeeChart components are printed, and describes how to improve printing.

Introduction

The TeeChart library is sending Charts to the printer using a Windows graphic format known as "metafile". This format has several advantages and disadvantages over using the bitmap format:

- It's much smaller, because graphic information is stored in vector instructions, instead of pixels. This results in a faster printing speed in most cases.

- The printer doesn't need to have more memory to print 'large' images (Charts).

- Due to be a sequence of instructions instead of a fixed array of pixels, a metafile can be resized (or "stretched") without losing resolution, and with greater accuracy.

- New printers support metafile format natively, via software driver or via "gdi hardware".

With Windows 32-bit (Windows 95 and Windows NT), an extended metafile format is implemented. This is known as "enhanced metafile". Metafiles have the extension: "*.wmf" and enhanced metafiles: "*.emf".

Design issues

The main goal of using metafiles in TeeChart, is to provide "wysiwyg" (what you see is what you get). That is, printed Charts should look as close as possible to how they look at screen.

To do this, TeeChart creates a metafile of the Chart image, and then sends this metafile to the printer. At this point, the Windows GDI module and the Windows Printer driver will take care of "painting" the Chart image on paper. This involves "stretching" or "resizing" the metafile, from screen coordinates to printer logical pixels.

Example:

A Chart on screen is at rectangle:

Left : 100

Top : 100

Right : 300

Bottom: 400

And it gets printed at this paper rectangle:

Left : 400

Top : 1000

Right : 1200

Bottom: 1500

In this example, the Chart should be rescaled both

in width and height:

screen Width = 300 - 100 = 200

paper Width = 1200 - 400 = 800

relation screen / paper = 800 / 200 = 4 <---

screen Height = 400 - 100 = 300

paper Height = 1500 - 1000 = 500

relation screen / paper = 500 / 300 = 1.666... <---

The problem is the relation between the horizontal and vertical increments is not the same:

4 <> 1.666...

This means the Chart will be expanded more in the horizontal dimension than in the vertical dimension. In this case, Windows increases font sizes and pen widths using the new dimensions.

Note: Windows 16-bit (3.1, 3.11) do not resize fonts as it does Windows 32-bit (95 and NT).

Text can overlap other Chart sections and look bad positioned. To fix this, see below "How to print proportionally?"

Windows 32-bit does a better job using the enhanced metafile format, so text dimensions are calculated precisely.

How to Print Proportionally?

Some printers can show problems when printing Charts *without* using proportional printer margins. Specially in Windows 16-bit with Delphi 1.0

The attached code can be used generically in your applications to calculate the optimal printer

printer margins so the relation between Chart Width and Height on screen is the same as on paper.

Another approach is to resize the Chart onscreen to match the proportion of the printer paper:

```
Chart1.Height:=Round(1.0*Printer.PageHeight*Chart1.Width/Printer.PageWidth);
Chart1.Print;
```

Other problems

Increasing resolution:

The metafile format is not aware of the "resolution" concept. This is means resolution information is not stored inside the metafile image. You can modify the Chart "resolution" *before* printing,

by setting this property:

```
Chart1.PrintResolution := -100 ;
```

Negative values (like -100 above), represent the porcentual increment in resolution. Zero means "wysiwyg".

More resolution is obtained by making all Chart fonts smaller and thinner lines. The metafile size is bigger when using more resolution. Increasing resolution can improve inaccuracy in buggy printer drivers calculations.

Printing non-solid lines:

In some printer / windows combinations, non-solid pen lines such as "dot" or "dash" will be printed as solid. This happens specially in some HP Laserjet printers. It seems the only workaround is to set the Chart Pen Width properties to Zero:

```
Chart1.LeftAxis.Grid.Width := 0;
```

Increasing resolution (see above) can make the printer to show non-solid lines.

Colors:

Many printers accept only a subset of the available Colors. This means setting a Chart color to a non-supported palette color may result in that color not being used by the printer, thus not drawing anything. In this situations, try with "well known" solid colors like "clRed, clBlue, clYellow, clGreen".

Some printers include a "Color Mapping" configuration dialog. (At Printer Properties dialog).

Printing directly:

Drawing a Chart directly onto a Printer GDI Handle or Canvas is also possible. The following code does it:

```
Uses Printers;
With Printer do
begin
 BeginDoc;
 try
  Chart1.Draw( Canvas, Rect( 0,0,PageWidth,PageHeight ) );
 finally
  EndDoc;
 end;
end;
```

You will see several problems printing directly:

-- The Chart background is gray color (instead of white).

-- Font sizes are extremely small.

-- There are many axis grid lines.

-- Lines are very thin.

One way to solve the above problems is using the metafile printing mode, described in this document. Another way (much more complicated) is to change all Font sizes and pen Width properties.

Printer driver settings:

Try to use always the latest "good" printer driver version. Try changing the Windows Printer driver resolution settings, and the spooler method (in Windows NT) to both "EMF" and "RAW" modes. "EMF" means all output is sent to the printer in metafile format.

Printing Reference

See the help file for extended information and examples on the following properties and methods. The BASIC.PAS and UPRINT.PAS units in TeeDemo project contain code showing custom printing.

The attached project shows Printing Margin adjustments to avoid text overlaping in Delphi 1.0 on 16-bit Windows systems.

Printing Properties:

Properties involved in Chart printing are:

Chart1.PrintMargins

The percentual space at the four sides of the paper page.

Chart1.PrintResolution

The relation between screen dimensions and paper dimensions.

Printing methods:

The TeeChart control has several methods designed for printing:

*Methods that print *and* eject the printed page:*

These are the default and more used TeeChart printing methods.

Chart1.Print

Uses default paper orientation and margins.

Chart1.PrintPortrait

Sets paper in Portrait and uses default margins.

Chart1.PrintLandscape

Sets paper in Landscape and uses default margins.

Chart1.PrintRect

Uses default paper orientation and the Rect parameter to position the Chart on the page.

Methods that do not eject the printed page:

These methods allow printing more than one Chart in the same paper page, or print other things and Chart components on the same page.

With these methods you need to call Printer.BeginDoc and EndDoc yourself. See Delphi TPrinter documentation and the UPRINT.PAS and BASIC.PAS units in TeeChart TEEDEMO project.

Chart1.PrintPartial

Draws a Chart to the Printer Canvas at the passed rectangle.

Chart1.PrintPartialCanvas

Draws a Chart to the passed Canvas at the passed rectangle.

Related methods and properties:

Not directly involved with printing, but useful for advanced printing:

Chart1.Draw

Draws a Chart to the passed Canvas, in screen "mode".

"Screen mode" means with gray background and without

using the metafile format.

This method draws directly to the Canvas.

Chart1.TeeCreateMetafile

Function that returns a metafile image of the Chart,

with the passed Rect coordinates.

Chart1.Metafiling

Boolean property indicating the Chart is now drawing

onto a metafile image.

Chart1.Printing

Boolean property indicating the Chart is now drawing

onto a Printer Canvas.

More information:

Information about printing and metafiles can be found at Microsoft Windows 32-bit SDK help file, located at:

"..\Delphi 5.0\Help\Win32.hlp"

Use the "Help Contents" (not the index) and scroll-down up to the Metafiles chapter.

See the Delphi's TMetafile and TMetafileCanvas objects at Delphi's help file and GRAPHICS.PAS unit.

Chart Zoom and Scroll

Scrolling and Zooming a Chart is simply setting its Axis scales to the desired values. After zooming or scrolling a Chart, all Series will repaint their points in their new positions.

Note: Pie Series can't be scrolled or zoomed. You can control Pie dimensions using Chart margins or Pie custom radius properties.

Topics in this section:

Zoom

Animated Zoom

Zooming by code

Undoing Zoom

Zoom Events

Scrolling

Scroll event

Controlling scroll

Keyboard Scrolling

Zoom

Charts can be zoomed programmatically or by user interaction with mouse dragging. The Chart AllowZoom property controls if users can apply zoom:

```
Chart1.AllowZoom := True ;
```

Users can zoom drawing a rectangle around the Chart area they want to see in detail.

Note:

Dragging should be done from top / left to bottom down. Dragging in the opposite direction resets axis scales (no zoom).

You can decide which mouse button and / or keys must be pressed to draw the zoomed area rectangle. The following code uses **global variables:**

```
TeeZoomMouseButton  := mbLeft ; { left mouse button used to zoom }
TeeZoomKeyShift   := [ ssShift ] ; { SHIFT key should be pressed to start
  zoom }
```

As soon as users release the mouse button, TeeChart repaints to show the zoomed area.

Animated Zoom

You can control if TeeChart will calculate zoom positions immediately or it will be calculating zoom in short "steps" until reaching the desired zoom window.

This makes an "animated" zoom effect, which helps to identify better the zoomed area. This code activates animated zoom:

```
Chart1.AnimatedZoom:= True ;
```

Set the AnimatedZoomSteps property to the desired number of intermediate zooms:

```
Chart1.AnimatedZoomSteps:= 5 ;
```

Advanced:

You can control how linear the animated zoom steps are, using the **global variable** AnimatedZoomFactor, from 1 to n:

```
AnimatedZoomFactor:=2.0;
```

Zooming by code

You can zoom in or zoom out a Chart using any of these methods:

ZoomRect adjusts axis scales to show the TRect parameter area. The rectangle is expressed in screen pixel co-ordinates. Rectangle areas inside Chart1.ChartRect rectangle will zoom in, while area outside ChartRect will zoom out.

```
Chart1.ZoomRect( Rect( 100, 100, 200, 200 ) );
```

ZoomPercent sets Chart zoom to a percentual value. This example sets Zoom to 110 % percent to zoom in :

```
Chart1.ZoomPercent( 110 );
```

To reset zoom:

```
Chart1.ZoomPercent( 100 );
```

The above methods work independently, you can use both at the same time.

Undoing Zoom

The UndoZoom method resets axis scales to their automatic Minimum and Maximum values:

```
Chart1.UndoZoom;
```

This will display all Series points, undoing any previous zoom in or zoom out operation, either by code or using the mouse.

Note:

If you want axis scales to be at specific values after undoing zoom, you can use the Chart OnUndoZoom event, documented below.

The Zoomed boolean property returns if all four Chart Axis are automatic or not. If not Chart1.Zoomed then Chart1.ZoomPercent(150);

Zoom Events

The OnZoom event is triggered whenever zoom is applied to a Chart, either manually or programmatically:

```
procedure TForm1.Chart1Zoom(Sender: TObject);
begin
    Button1.Visible:=True ;  { make visible the
            “no-zoom” button }
end;
```

The OnUndoZoom event is called when undoing zoom, by code or by mouse.

Scrolling

Scrolling is very similar to zoom. Axis scales are incremented or decremented and the whole Chart component is repainted to show Series points at their new positions.

The Chart AllowPanning property controls if users can scroll Chart contents by dragging the mouse. Its possible values are:

pmNone	No scroll is allowed.
pmHorizontal,	
pmVertical	Allow scroll only on these directions.
pmBoth	Allow complete scroll over all directions.

Example:

```
Chart1.AllowPanning := pmNone ; { no scroll is
         allowed }
```

Like Zoom, the following **global variables** control mouse and keyboard requirements to start scrolling:

```
TeeScrollMouseButton := mbRight; { button used to scroll }
TeeScrollKeyShift  : [ ssCtrl ]; { CONTROL key should be pressed to start
  scroll }
```

You can programmatically scroll a Chart, using the Axis Scroll method:

```
Procedure Scroll(Const Offset:Double;
           CheckLimits:Boolean);
```

Example:

```
Chart1.BottomAxis.Scroll( 1000, True );
```

The above code increments BottomAxis scales by 1000. This is the same as doing:

```
With Chart1.BottomAxis do SetMinMax( Minimum+1000, Maximum+1000 );
```

and setting BottomAxis Automatic property to False.

The Chart will repaint and the horizontal bottom axis will be “scrolled” to the left a quantity of “1000” in axis scales.

The “CheckLimits” boolean parameter instructs the axis to scroll ONLY if there are more Series points in the scrolling direction.

Scroll event

The Chart OnScroll event is fired every time users scroll manually the Chart.

```
procedure TForm1.Chart1Scroll(Sender: TObject);
begin
  Label1.Caption := ‘This Chart has scrolled ! ‘ ;
end;
```

Controlling scroll

The OnAllowScroll event can be used to programmatically accept or refuse a planned scroll:

```
procedure TForm1.Chart1AllowScroll(Sender: TChartAxis; var AMin,
 AMax: Double; var AllowScroll: Boolean);
begin
 if Sender = Chart1.BottomAxis then
   if AMax > 1000 then AllowScroll := False ;
end;
```

The above code refuses user scrolling if attempting to set bottom axis maximum to a value greater than 1000. The same checking can be performed over the DateTime axis:

```
if Sender = Chart1.BottomAxis then
   if AMax > EncodeDate( 1997, 12, 31 ) then AllowScroll := False ;
```

Keyboard Scrolling

The UKEYBOA.PAS example unit shows how to use the Form OnKeyDown event and scroll when pressing the arrow keys. First, the Form KeyPreview property should be set to True.

At the KeyDown event, and depending on the pressed arrow key, the four Chart axis are scrolled using the Axis Scroll method.

The example also uses ZoomPercent and UndoZoom methods.

Real-Time Charting and Speed

Two big rules apply to speed performance in real-time Charting:

1. Plot as few points as possible

2. Use the fastest possible hardware.

Together, these two rules really make a big speed difference when drawing Charts many times continuously.

Some other suggestions are:

- Use 2D. Three dimensional Charts are slower to paint than 2D.

- Make Charts small. Bigger Charts need more pixels to be filled.

- Remove Chart Legend and Titles when possible.

- Use default fonts and font sizes.

- Use FastLineSeries for fastest way to plot many points.

- Use solid pens and brushes styles.

- Avoid using circular shapes or circular bar styles.

- Don’t use background bitmaps or gradient fill effects.

- Set Chart BevelInner and BevelOuter properties to bvNone

- When possible, set Chart1.AxisVisible to False to remove axis.

- Set your video mode resolution and colour depth to the optimum values, according to your video card.

- A combination of 800x600 x 256 colours can be faster than 1024x768 x 32k colours, on average video cards.

- Use Windows 95 or Windows NT with accelerated drivers for your video card.

BufferedDisplay

This public property controls how Charts are internally drawn. When True, the default, Charts are drawn into an internal hidden bitmap. When drawing is finished, this internal bitmap is transferred to the screen in one single operation. The result is non-flickering animation.

In some circumstances, setting BufferedDisplay to False can accelerate Chart redrawing. It depends on CPU and video card speed, and on the number of points plotted.

The bad news is that drawing directly to screen produces flickering, but can be helpful for really heavy redrawing applications.

The UFAST.PAS example unit can be used as a benchmark for testing drawing speed times. Remember to use exactly the same data values in your tests with your real Charts. Use always the same number of points and the same point values when comparing speed results. Run the tests many times to obtain a fair statistical result and restart completely between tests.

Functions

A function may be defined to use one of most Series types as a ‘carrier’ Series and to act upon other Series to create its source data. When the function is defined it behaves as any other Series. Functions may thus be built upon other functions. Only in its definition of data source is a function different from other Series as it contains information that relates it statistically to other data sources by formula, sum, etc.

Topics in this Section:

Adding a function with the Chart editor

Functions may be added using the Chart editor at design time. Similarly to adding a Series you may use the TeeChart Gallery to choose the function. In the Gallery all functions are initially displayed as LineSeries functions - you may change Series type after initially adding the function.

When you have added the function go to the DataSource tab of the Series page. Here you may choose which input Series (or Series) to add to the function. The first Series in the selected listbox, reading from top to bottom, is the Series on which the operation is performed in the case of function which doesn’t treat each member of the list homogeneously.

Eg. **Subtract**.

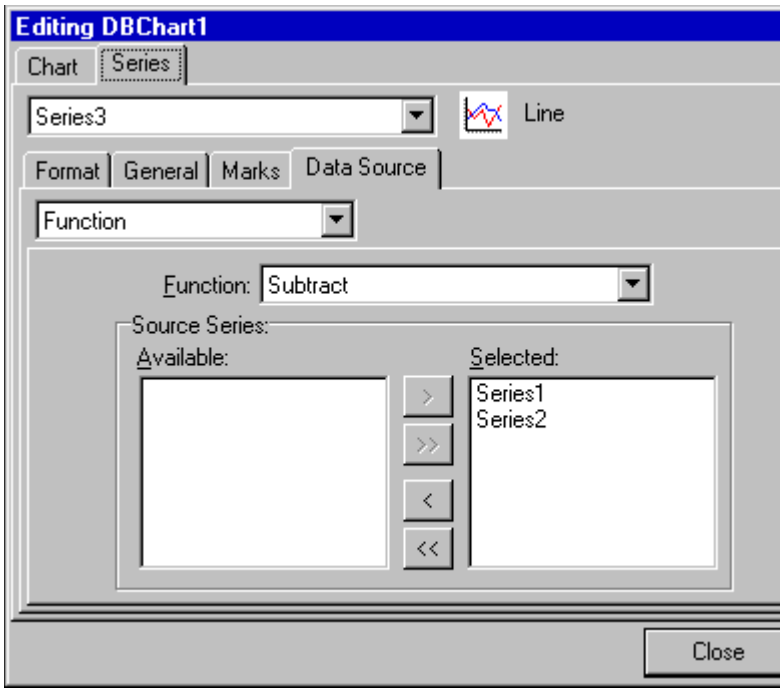


Fig. 1. Subtract function defined in the Chart editor

In the Chart editor screen you see Series3 being defined with inputs Series1 and Series2. The Series order in the list defines which Series is doing the subtraction.

Here: *Series3:= Series1 - Series2*

Deleting a function with the Chart editor

Alternatively you may delete the Series you have added as carrier for the function (delete Series from the first page of the Chart editor) or you may redefine the Series as having a different datasource in the DataSource tab of the Series page.

Changing a FunctionType with the Chart editor

The option to change the FunctionType is in the DataSource tab of the Series page for the function Series. The drop down combobox contains a list of all function types - You may choose any from the list. Copy will directly copy the input Series (duplicating the Series).

Adding a function by code

Function is a component. When you add a new function you are adding a Series, defining a new function and setting it as FunctionType for the Series.

```
Series1.SetFunction(TAddTeeFunction.Create(Self));
```

See the online help for a description of how to add each of the different types of function. Each function uses the same Series method, SetFunction.

Deleting a function by code

To delete a function from a Series either delete the Series or use the SetFunction method to de-allocate the connection between the Series and the Function.

```
Series1.SetFunction(nil);
```

Period

You may find the Period property extremely useful when working with functions. It is used to define the frequency for which the function re-calculates.

Example

We have 6 data points (eg. bars of a Bar Series) with values:

3, 8, 6, 2, 9 and 12

We define a function Series with Period 0 (default when only one Series is input to the Function) the average drawn is:

6.667

With Period set to 2 we get 3 values of average as output from the function:

5.5, 4 and 10.5

These values will plot centrally in their period range, ie. The 1st value between bars 1 and 2 of the input Series, 2nd value between bars 3 and 4, etc..

You may define Period by selecting the function in the Object Inspector or you may modify Period at runtime using FunctionType.

Eg. Where Series 2 is the function Series:

```
Series2.FunctionType.Period:=2
```

TeeChart examples

The TeeChart folder below the Delphi Demos folder contains some TeeChart examples with source code.

For a further look at coded examples please take a look at the TeeChart website, www.teechart.com, which contains many additional technical examples and Frequently Asked Questions.

Calculate3DPosition Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration

```
function Calculate3DPosition (x,y,z:Integer):TPoint; virtual; abstract;
```

Description

Calculates and returns the XY position in pixels of the XYZ 3D coordinate.

Can be used when custom drawing using 3D XYZ coordinates, either returned from the axes or not.

Example

```
Chart1.OnAfterDraw...
Var P:TPoint;
With Chart1.Canvas do
begin
 P:=Calculate3DPosition( ChartXCenter, ChartYCenter, 20 );
 Cube( P.X-10,P.X+10,P.Y-10,P.Y+10,15,25, True );
end;
Draws a Cube centered on the middle of the Chart, at Z position of "20".
```

Arrow Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure Arrow ( Filled:Boolean; Const FromPoint,ToPoint:TPoint;
ArrowWidth,ArrowHeight,Z:Integer); virtual; abstract;
```

Description

Draws a line with an arrow head of ArrowWidth and ArrowHeight dimensions in pixels.

Cube Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure Cube (Left,Right,Top,Bottom,Z0,Z1:Integer; DarkSides:Boolean);
virtual; abstract;
```

Description

Draws a Cube, with optionally shadowed top and right sides.

Cylinder Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure Cylinder (Vertical:Boolean; Left,Top,Right,Bottom,Z0,Z1:Integer;
DarkCover:Boolean); virtual; abstract;
```

Description

Draws cylinder toggle Boolean for vertical or horizontal cylinder.

HorizLine3D Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration
```
procedure HorizLine3D (Left,Right,Y,Z:Integer); virtual; abstract;
```

Description
Line from (Left,Y,Z) to (Right,Y,Z)

VertLine3D Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration
```
procedure VertLine3D (X,Top,Bottom,Z:Integer); virtual; abstract;
```

Description
Line from (X,Top,Z) to (X,Bottom,Z)

ZLine3D Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration
```
procedure ZLine3D (X,Y,Z0,Z1:Integer); virtual; abstract;
```

Description
Line from (X,Y,Z0) to (X,Y,Z1)

EllipseWithZ Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration
```
procedure EllipseWithZ (X1, Y1, X2, Y2, Z: Integer); virtual; abstract;
```

Description
Ellipse bounding Rect (X1,Y1,X2,Y2) at Z position.

FrontPlaneBegin Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure FrontPlaneBegin ; virtual; abstract;
```

Description

Disables rotation, elevation, offsets and zoom.

The Legend, Title and Foot use this.

FrontPlaneEnd Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration
```
procedure FrontPlaneEnd ; virtual; abstract;
```

Description
Enables rotation, elevation, offsets and zoom again. The Legend, Title and Foot use this.

LineWithZ Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration
```
procedure LineWithZ (X0,Y0,X1,Y1,Z:Integer); virtual; abstract;
```

Description
Line from (X0,Y0,Z) to (X1,Y1,Z)

LineTo3D Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration
```
procedure LineTo3D (X,Y,Z:Integer); virtual; abstract;
```

Description
same as TCanvas , with z

MoveTo3D Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration
```
procedure MoveTo3D (X,Y,Z:Integer); virtual; abstract;
```

Description
same as TCanvas, with z

Pie3D Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure Pie3D ( XCenter,YCenter,XRadius,YRadius,Z0,Z1:Integer; Const
StartAngle,EndAngle:Double;  DarkSides,DrawSides:Boolean); virtual;
abstract;
```

Description

Draws a Pie slice ...

Plane3D Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration

```
procedure Plane3D (Const A,B:TPoint; Z0,Z1:Integer); virtual; abstract;
```

Description
Rectangle from (A.X,A.Y,Z0) to (B.X,B.Y,Z1)

PlaneWithZ Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration
```
procedure PlaneWithZ (P1,P2,P3,P4:TPoint; Z:Integer); virtual; abstract;
```

Description
Draws a polygon of (P1,P2,P3,P4) at Z position

PlaneFour3D Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration

```
procedure PlaneFour3D (Points:TPlaneFourPoints; Z0,Z1:Integer); virtual;
abstract;
```

Description
Draws a polygon of four points. The first and second point at Z0 position, the third and fourth point at Z1 position.

RectangleWithZ Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure RectangleWithZ (Const Rect:TRect; Z:Integer); virtual; abstract;
```

Description

Rectangle at Z position

Example

```
Chart1.Canvas.RectangleWithZ( Rect( 50,20,150,80), 15 );
```

RectangleY Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure RectangleY (Left,Top,Right,Z0,Z1:Integer); virtual; abstract;
```

Description

Horizontal Rectangle from Left to Right, from Z0 to Z1 position, at Top Y.

RectangleZ Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure RectangleZ (Left,Top,Bottom,Z0,Z1:Integer); virtual; abstract;
```

Description

Vertical Rectangle from Top to Bottom and from

Z0 to Z1 position, at Left X.

RotateLabel3D Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration

```
procedure RotateLabel3D (x,y,z:Integer; Const St:String; RotDegree:Integer);
virtual; abstract;
```

Description
Draws a 2D rotated label at XYZ position. Depends on TextAlign property.

TextOut3D Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure TextOut3D (X,Y,Z:Integer; const Text:String); virtual; abstract;
```

Description

Draws a 2D non-rotated label at XYZ position. Depends on TextAlign property.

TriangleWithZ Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure TriangleWithZ (Const P1,P2,P3:TPoint; Z:Integer); virtual;
abstract;
```

Description

Draws a triangle of P1,P2,P3 at Z position.

Example

```
Chart1.Canvas.TriangleWithZ( Point(40,40), Point(50,30), Point( 60, 40),
  10 );
```

Pixels3D Property (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
property Pixels3D [X,Y,Z:Integer]:TColor;
```

Description

Sets the 3D pixel to TColor, using Pen.Width (same as TCanvas)

Example

```
Pixels3D[ 100,50,30 ]:= clRed ;
```

Supports3DText Property (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
property Supports3DText :Boolean;
```

Description

Returns if Canvas can rotate text in 3D mode.

Only OpenGL Canvas can rotate text.

SupportsFullRotation Property (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
property SupportsFullRotation :Boolean;
```

Description

Returns if Canvas can do rotation and elevation of more than 90 degree. Only OpenGL Canvas has 360 degree rotation.

Example

```
if Chart1.Canvas.SupportsFullRotation then
  Chart1.View3DOptions.Rotation:= 195;
```

DoHorizLine Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure DoHorizLine (X0,X1,Y:Integer); virtual; abstract;
```

Description

Draws horizontal line at X0 to X1 at vertical position Y

DoRectangle Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure DoRectangle (Const Rect:TRect); virtual; abstract;
```

Description

The DoRectangle method calls the Chart.Canvas.Rectangle method passing the Rect variable parameter. It is equivalent to:

```
Canvas.Rectangle( Rect.Left, Rect.Top, Rect.Right, Rect.Bottom ) ;
```

DoVertLine Method (Canvas)

See also

Applies to
TCanvas3D component

Declaration
```
procedure DoVertLine (X,Y0,Y1:Integer); virtual; abstract;
```

Description
Draws vertical line at Y0 to Y1 at horizontal position X.

EraseBackground Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure EraseBackground (const Rect: TRect); virtual; abstract;
```

Description

Clears background defined for TRect (Rectangle(Rect.Left, Rect.Top, Rect.Right, Rect.Bottom)).

GradientFill Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure GradientFill ( Const Rect:TRect; StartColor,EndColor:TColor;
Direction:TGradientDirection); virtual; abstract;
```

Description

The GradientFill method is used to fill a Screen area with multi-colored lines to obtain a nice shadow effect and coloured backgrounds. The Chart.Gradient component uses this method internally.

Example

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Chart1.Canvas.GradientFill( Canvas, ClientRect, clYellow, clBlue, True);
end;
```

Line Method (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
procedure Line (X0,Y0,X1,Y1:Integer); virtual; abstract;
```

Description

Draws line between end co-ordinates using current Pen

RotateLabel Example

```
Chart1.RotateLabel ( 100, 100, 'Hello', 90 ) ;
```

RotateLabel Method (Canvas)

See also Example

Applies to

TCanvas3D component

Declaration

```
procedure RotateLabel (x,y:Integer; Const St:String; RotDegree:Integer);
virtual; abstract;
```

Description

This method will draw a rotated text string at the specified xy coordinates with the RotDegree rotation angle. RotDegree values must be between 0 and 360. The string is drawn on the Chart.Canvas.

BackColor Property (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
property BackColor :TColor;
```

Description

Sets / returns the color used to fill spaces when displaying text or filling with brushes with different style than bsSolid.

BackMode Property (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
property BackMode :TCanvasBackMode;
```

Description

Can be cbmOpaque or cbmTransparent.

If it is cbmTransparent, then the BackColor is used.

Canvas Property (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
property Canvas :TCanvas;
```

Description

Please refer to Delphi TCanvas. This property offers access to all properties and methods of Delphi's TCanvas.

Pen Property (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
property Pen :TPen;
```

Description

Please refer to Delphi TPen.

Brush Property (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
property Brush :TBrush;
```

Description

Please refer to Delphi TBrush.

Font Property (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
property Font :TFont;
```

Description

Please refer to Delphi TFont.

Metafiling Property (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
property Metafiling :Boolean;
```

Description

Returns if the Canvas is drawing to a Metafile Handle or to a TMetafileCanvas.

Monochrome Property (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
property Monochrome :Boolean;
```

Description

Sets the internal Bitmap.Monochrome property.

ReferenceCanvas Property (Canvas)

See also

Applies to
TCanvas3D component

Declaration
```
property ReferenceCanvas :TCanvas;
```

Description
The Delphi TCanvas associated to the Canvas3D. Can be used for low level tasks only.

Pyramid Property (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
property Pyramid :
(Vertical:Boolean;Left,Top,Right,Bottom,z0,z1:Integer;Darksides:Boolean);
Virtual; abstract;
```

Description

Draws a vettical or horizontal Pyramid with optional dark shaded sides.

TextAlign Property (Canvas)

See also

Applies to

TCanvas3D component

Declaration

```
property TextAlign :TCanvasTextAlign;
```

Description

Sets the alignment used when displaying text using TextOut or TextOut3D. See Delphi TAlignment.

UseBuffer Property (Canvas)

See also

Applies to
TCanvas3D component

Declaration
```
property UseBuffer :Boolean;
```

Description
Same as TChart.BufferedDisplay.

Only the normal TCanvas3D can do BufferedDisplay (UseBuffer:=True)

ExplodeBiggest Property (TPieSeries)

See also

Applies to
TPieSeries component

Declaration
```
property ExplodeBiggest :Integer;
```

Description
Use this property to explode out from the chart the largest slice.

OtherSlice Property (TPieSeries)

See also

Applies to

TPieSeries component

Declaration

```
property OtherSlice :TPieOtherslice;
```

Description

Use this property to define the grouping size for the 'Other' slice of the Pie. Grouping may be expressed as a percentage or value (see Otherslice.Style).

GridCentered Property (TChartAxis)

See also

Applies to
TChartAxis component

Declaration
```
property GridCentered :Boolean;
```

Description
Default False Places Axis Grid lines between label positions

Style Property (TPieOtherSlice)

See also

Applies to

TPieOtherSlice component

Declaration

```
property Style :TPieOtherStyle;
```

Description

Defines whether to use value or percentage to group the 'other' Pie slice.
TPieOtherStyle=(poNone,poBelowPercent,poBelowValue);

Text Property (TPieOtherSlice)

See also

Applies to
TPieOtherSlice component

Declaration
```
property Text :String;
```

Description
label for 'Otherslice'

Value Property (TPieOtherSlice)

See also

Applies to
TPieOtherSlice component

Declaration
```
property Value :Integer;
```

Description
Threshold value below which to include data point (or slice) in grouped 'Otherslice'.

TPieOtherSlice Component

Properties

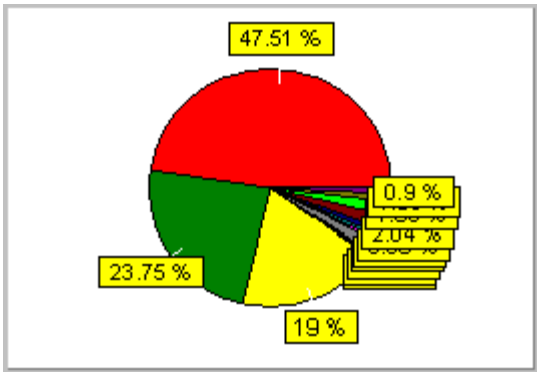
Unit
Series

Ancestor
TPersistent

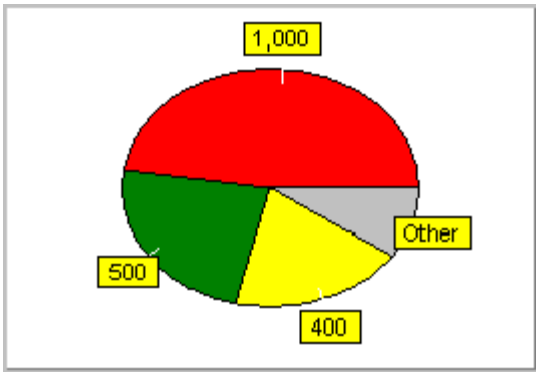
Description
The TPieOtherSlice component controls the grouping of Pie slices. Use the Style property to define how, by percentage or value, you wish to group the smaller value slices of the Pie.

Use of TPieOtherSlice:

Before applying OtherSlice



After applying OtherSlice



Properties

Run-time only

Key Properties

Style

Text

Value

See Also

TPieSeries.ExlodeBiggest

TPieOtherSlice.Style

TPieOtherSlice.Text

TPieOtherSlice.Value

See Also

TCanvas3D.Brush
TCanvas3D.Calculate3DPosition
TCanvas3D.Canvas
TCanvas3D.Font
TCanvas3D.FrontPlaneBegin
TCanvas3D.FrontPlaneEnd
TCanvas3D.Metafiling
TCanvas3D.Monochrome
TCanvas3D.Pen
TCanvas3D.ReferenceCanvas
TCanvas3D.UseBuffer
TCanvas3D.Plane3D
TCanvas3D.PlaneWithZ
TCanvas3D.PlaneFour3D

See Also

TCanvas3D.RotateLabel3D

TCanvas3D.SupportsFullRotation

TCanvas3D.TextOut3D

TCanvas3D.Supports3DText

TCanvas3D.RotateLabel

TCanvas3D.TextAlign

See Also

TCanvas3D.Arrow

TCanvas3D.HorizLine3D

TCanvas3D.VertLine3D

TCanvas3D.ZLine3D

TCanvas3D.EllipseWithZ

TCanvas3D.LineWithZ

TCanvas3D.LineTo3D

TCanvas3D.MoveTo3D

TCanvas3D.Line

TCanvas3D.DoHorizLine

TCanvas3D.DoVertLine

See Also

TCanvas3D.BackColor

TCanvas3D.BackMode

TCanvas3D.Cube

TCanvas3D.Cylinder

TCanvas3D.Pie3D

TCanvas3D.RectangleWithZ

TCanvas3D.RectangleY

TCanvas3D.RectangleZ

TCanvas3D.TriangleWithZ

TCanvas3D.Pixels3D

TCanvas3D.DoRectangle

TCanvas3D.GradientFill

TCanvas3D.EraseBackground

Clone Series

Description
(Chart Editor) Makes a new, duplicate Series of selected Series.

Class hierarchy

This listing shows the relationship between TeeChart Classes and their descended classes.

- AxisException
- BarException
- ChartException
    - PieException
- DBChartException
- LegendException
- TAreaSeriesEditor
- TArrowSeriesEditor
- TAxisIncrement
- TAxisMaxMin
- TBarSeriesEditor
- TBrushDialog
- TCandleEditor
- TChartBrush
- TChartClassProperty
    - TChartBrushProperty
    - TChartPenProperty
- TChartCompEditor
    - TDBChartCompEditor
- TChartEditForm
- TChartFontObject
    - TChartAxisTitle
    - TChartTitle
    - TCustomChartLegend
        - TChartLegend
- TChartListBox
- TChartPen
    - TChartArrowPen
    - TChartAxisPen
    - TChartHiddenPen
    - TDarkGrayPen
    - TDottedGrayPen
- TChartPreview
- TChartScrollBar
- TChartSeries
    - TChartShape
    - TCircledSeries
        - TCustomPolarSeries

- TClockSeries
- TPolarSeries
- TRadarSeries
- TWindRoseSeries
- TPieSeries
- TCursorSeries
- TCustom3DSeries
- TCustom3DGridSeries
- TContourSeries
- TSurfaceSeries
- TPoint3DSeries
- TCustomBarSeries
- TBarSeries
- TBar3DSeries
- TCustomErrorSeries
- TErrorBarSeries
- TErrorSeries
- TImageBarSeries
- THorizBarSeries
- TCustomSeries
- TAreaSeries
- TBezierSeries
- TLineSeries
- TOHLCSeries
- TCandleSeries
- TBigCandleSeries
- TPointSeries
- TArrowSeries
- TBubbleSeries
- TCustomImagePointSeries
- TDeltaPointSeries
- TImagePointSeries
- TGanttSeries
- TMyPointSeries
- TVolumeSeries
- TFastLineSeries
- TChartSeriesList
- TChartShapeEditor
- TChartValueList
- TChartValueLists

- TChartWall
- TContourSeriesEditor
- TCustomChartAxis
    - TChartAxis
    - TChartDepthAxis
- TCustomChartEditor
    - TChartEditor
    - TChartPreviewer
- TCustomSeriesEditor
- TCustomTeeCommander
    - TTeeCommander
- TCustomTeeGradient
    - TChartGradient
- TCustomTeePanel
    - TCustomAxisPanel
        - TCustomChart
            - TChart
            - TCustomDBChart
                - TDBChart
                - TQRDBChart
    - TDraw3D
- TErrorSeriesEditor
- TExplodedSlices
- TFastLineSeriesEditor
- TFormPeriod
- TFormTee3D
- TFormTeeAxis
- TFormTeeGeneral
- TFormTeeLegend
- TFormTeePage
- TFormTeePanel
- TFormTeeSeries
- TFormTeeTitle
- TFormTeeWall
- TGanttSeriesEditor
- TGLLight
- TGLPosition
- TGrid3DSeriesEditor
- TImageBarSeriesEditor
- TListBoxSections

- TListOfDataSources
- TPenDialog
- TPieAngle
- TPieAngles
- TPieOtherSlice
- TPieSeriesEditor
- TPoint3DSeriesEditor
- TPolarSeriesEditor
- TQRChart
- TSeriesDataSet
- TSeriesMarkPosition
- TSeriesMarks
- TSeriesMarksPositions
- TSeriesPointer
- TSeriesPointerEditor
- TSurfaceSeriesEditor
- TTeeAboutForm
- TTeeCanvas
    - TCanvas3D
        - TGLCanvas
        - TTeeCanvas3D
- TTeeChartWizard
- TTeeDlgWizard
- TTeeDragObject
- TTeeExportForm
- TTeeFunction
    - TAddTeeFunction
    - TAverageTeeFunction
    - TBasicTeeFunction
    - TCountTeeFunction
    - TCumulative
    - TCustomFittingFunction
        - TCurveFittingFunction
    - TDivideTeeFunction
    - THighTeeFunction
    - TLowTeeFunction
    - TMovingTeeFunction
        - TExpAverageFunction
        - TMomentumFunction
        - TMovingAverageFunction

        - TRSIFunction
    - TMultiplyTeeFunction
    - TStdDeviationFunction
    - TSubtractTeeFunction
    - TTrendFunction
- TTeeGallery
- TTeeGalleryPanel
- TTeeOpenGL
- TTeePreviewPage
- TTeeSeriesType
- TTeeSeriesTypes
- TTeeTabControl
- TTreeCompEditor
- TUpDown
- TView3DOptions
- TVolumeSeriesEditor
- TZoomPanningRecord

SizeTickAxis Method (TChartAxis)

See also

Applies to

TChartAxis component

Declaration

```
Function SizeTickAxis :Integer;
```

Description

Returns the amount in pixels used by the axis line.

SizeTitle Method (TChartAxis)

See also

Applies to

TChartAxis component

Declaration

```
Function SizeTitle :Integer;
```

Description

Returns the amount in pixels used by the axis Title text.

SizeLabels Method (TChartAxis)

See also

Applies to

TChartAxis component

Declaration

```
Function SizeLabels :Integer;
```

Description

Returns the amount in pixels used by the axis labels text.

DrawAxisLabel Method (TChartAxis)

See also

Applies to
TChartAxis component

Declaration

```
procedure DrawAxisLabel (x,y,Angle:Integer; Const St:String);
```

Description
Draws Axis Label (String) at specified X,Y co-ordinate at Angle.

CalcPosValue Method (TChartAxis)

See also

Applies to
TChartAxis component

Declaration

```
Function CalcPosValue (Const Value:Double):Longint;
```

Description
Returns the coordinate position in pixels corresponding to the "Value" parameter in axis scales.

CalcSizeValue Method (TChartAxis)

See also

Applies to

TChartAxis component

Declaration

```
Function CalcSizeValue (Const Value:Double):Longint;
```

Description

Returns the amount in pixels that corresponds to a portion of the axis of size "Value" in axis scales.

CalcMinMax Method (TChartAxis)

See also

Applies to

TChartAxis component

Declaration

```
Procedure CalcMinMax (Var AMin,AMax:Double);
```

Description

Returns the minimum and maximum values of the associated Series.

CalcIncrement Method (TChartAxis)

See also

Applies to
TChartAxis component

Declaration

```
Function CalcIncrement :Double;
```

Description
When Increment property is zero, this function returns the best "Increment" value for axis labels that will make no labels overlap.

IStartPos Variable (TChartAxis)

See also

Applies to

TChartAxis component

Declaration

```
var IStartPos : Integer;
```

Description

Read-only. Returns the position in pixels for the start point of the axis.

It is calculated based on the StartPosition property.

IEndPos Variable (TChartAxis)

See also

Applies to

TChartAxis component

Declaration

```
var IEndPos : Integer;
```

Description

Read-only. Returns the position in pixels for the ending point of the axis.

It is calculated based on the EndPosition property.

IsDepthAxis Variable (TChartAxis)

See also

Applies to
TChartAxis component

Declaration

```
var IsDepthAxis : Boolean;
```

Description
Read-Only. Returns True when the axis object is the Depth
axis of a Chart. (ie: When it's equal to the Chart1.DepthAxis property).

See Also

TChartAxis.CalcPosValue
TChartAxis.CalcSizeValue
TChartAxis.CalcMinMax
TChartAxis.CalcIncrement
TChartAxis.IStartPos
TChartAxis.IEndPos
TChartAxis.IsDepthAxis
TChartAxis.CustomDraw
TChartAxis.Increment
TChartAxis.LabelsSize
TChartAxis.Maximum
TChartAxis.Minimum
TChartAxis.PosAxis
TChartAxis.PosLabels
TChartAxis.PosTitle
TChartAxis.TitleSize

BevelOuter property (TChart)

Applies to
TChart component

Declaration
```
property BevelOuter : Integer;
```

Description
Sets the size of the bevel that will form the outer side of the Chart panel.

BevelInner property (TChart)

Applies to
TChart component

Declaration
```
property BevelInner : Integer;
```

Description
Sets the size of the bevel that will form the inner side of the Chart panel.

Chart editor

See also

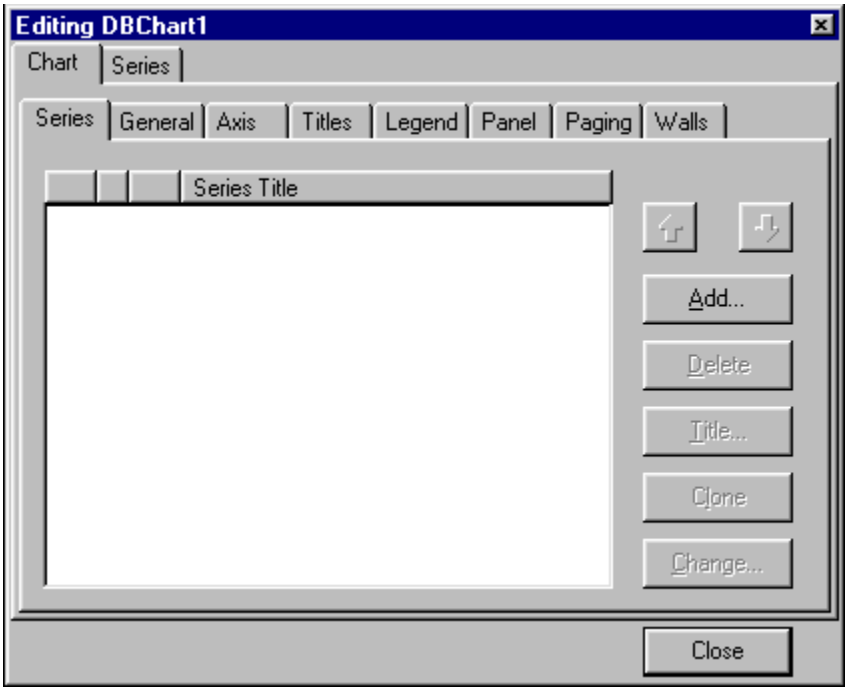
Description

The Chart editor presents the means to configure both chart and data series.

Design-time only in TeeChart 4 runtime version. To access the Chart editor double-click on any TChart or TDBChart component placed on a Form. Alternatively right mouse-click on the TChart to bring up a menu which includes **Chart editor** as one of the options.

To edit a TQRChart you may follow the same procedure as above whilst the TQRChart is placed on a QuSoft QuickReport.

See the user guide for more information about the Chart Editor.



TeeChart version 4 Help and information resources

TEECHART.HLP

In addition to covering all TeeChart Classes, properties, events, and methods, TeeChart's online Help system provides a handy Quick Start section.

To view Help, press F1 when your cursor is on any TChart code element, property, or object. You can also open the main Help system through the Help menu and view the TeeChart Help Contents or search the Index for TeeChart elements, or open TEECHART.HLP directly (and independently) from Windows Explorer.

Example code

TeeChart provides a number of code examples in the folder \Demos\TeeChart below your Delphi installation directory.

teeMach Web pages

Additional example code, technical information and product news is always available on the Web at and information of interest are continually added to the web, http://www.teechart.com. For technical information, click the Technical link on the main product page.

teeMach email support

teeMach offer a paid email support service. For non-paid support we will reply to new issues that may be of interest to all to be placed on the teeMach web and included in the TeeChart FAQ. See the the teeMach web for details about obtaining support.

Newsgroups

Borland offer a newsgroup forum to discuss Charting and Reporting issues. Connect to the Newsgroup server FORUMS.BORLAND.COM and point your Internet News browser to borland.public.delphi.reporting-charting.

The TeeChart mailing list

This is an independant TeeChart mailing list maintained by Mr. Phil Scadden of the Institute of Geological and Nuclear Sciences of New Zealand. You should subscribe by sending a mail with the word SUBSCRIBE in the email message body (not the subject) to teechart-request@lhn.gns.cri.nz.