TMS VCL Chart
DEVELOPERS GUIDE

May 2017

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

Index

TMS VCL Chart availability

TMS VCL Chart is available as VCL component set for Win32/Win64 application development and can be used as well with VCL for the Web (IntraWeb)

VCL versions:
TMS VCL Chart is available for Delphi 7, 2007, 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin, C++Builder 2007, 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin (Prof/Enterprise/Architect)

TMS Advanced IntraWeb Charts is available for Delphi 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin C++Builder 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin (Prof/Enterprise/Architect)

TMS VCL Chart use

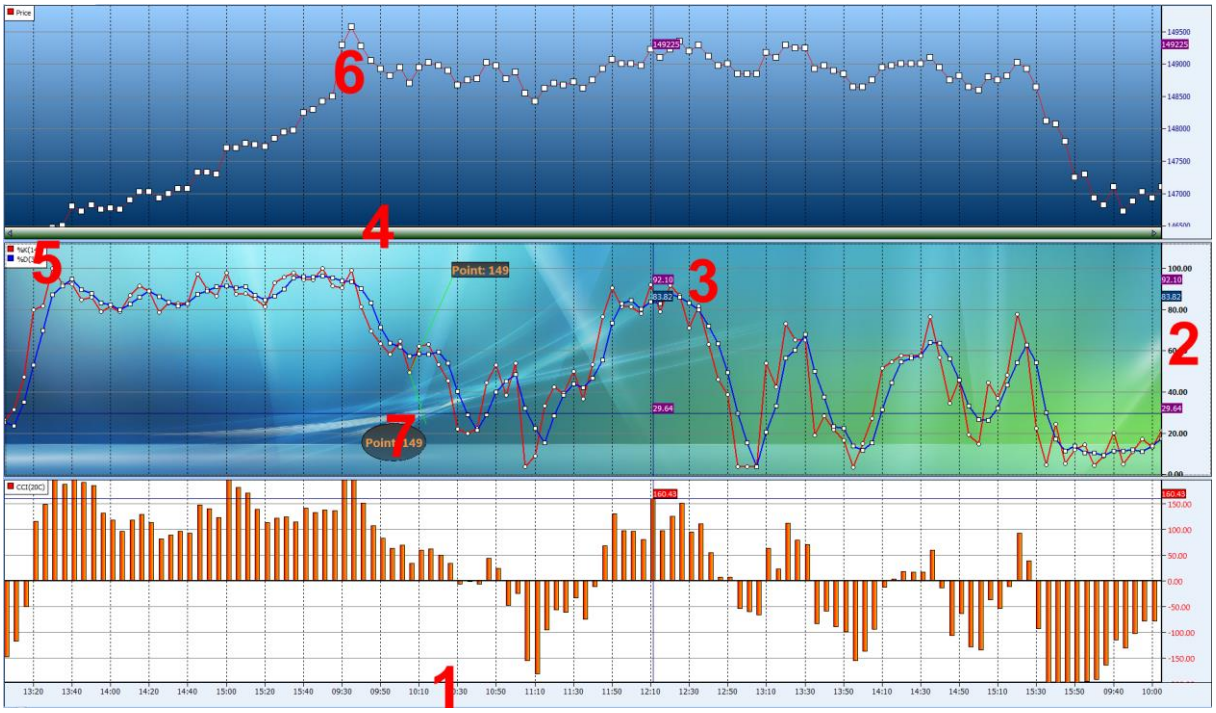
The TMS VCL Chart is designed to display different kinds of data. From financial and marketing data to monthly business sales, graphical and educative math data. The graphical user interface supports chart types such as bar, histogram, area, line, ohlc, candlestick, and many variants like stacked bar, stacked area, stacked percentage area, stacked percentage bar, pie charts, donut chart, spider chart, bubble chart … The line, bar or area charts can be shown in vertical and in horizontal direction. TMS VCL Chart is easy to use with features like multiple chart panes, scrolling, scaling, synched chart panes, synched crosshairs and much more…

TMS VCL Chart organisation

TMS VCL Chart is a component set which includes components such as TAdvChartView, TDBAdvChartView, TAdvChartViewGDIP (GDI +), TDBAdvChartViewGDIP (GDI+), TAdvChartLink, TAdvChartTypeSelector, and TMS VCL Chart editors to edit panes, series and annotations.

TAdvChartView

TAdvChartView consists of multipane charts. This means that the chartview can have one or more panes and each pane can display a chart with a single or multiple series. The charts on the panes can scroll and zoom synchronously or can also do this independent of each other. The panes of the chartview are accessible through the component's Pane collection.
The major elements of the chart are indicated on this screenshot:



1: X-axis:
Displays the range of points in the chart in number format, a date/time format with unit types minute, day, month, year, custom drawn values or specified values per point. The X-axis supports scaling with mouse. Include poHorizontalScaling in Pane.Options property, then click and drag mouse left or right to see more or less points.

2: Y-axis:
Displays the range of series values from a defined minimum to maximum or via Autorange the best range can automatically be choosen. The Y-axis supports scaling with the mouse and/or keyboard. Include poVerticalScaling in Pane.Options property, then click and drag mouse up or down or press Shift Up/Down to expand or reduce the maximum and minimum value. The Y-axis can be set at the left side of the chart, the right side of the chart or both sides. Different Y-axis values can be shown for different series. The Y-axis also has the capability to show major & minor units with a different font.

3: Crosshairs:
When crosshairs are enabled, move the mouse in the pane area and values which intersect with the crosshair are shown either in the Y-axis area, at the crosshair intersection point, in a separate tracker window or the values can be programmatically retrieved to display in another control.

4: Navigator:
Enable the navigator to scroll left or right in the chart pane. The navigator also offers the same capability as the X-axis itself to zoom in/out.

5: Legend:
Displays the legend text for the number of current chart series added to the chart pane.

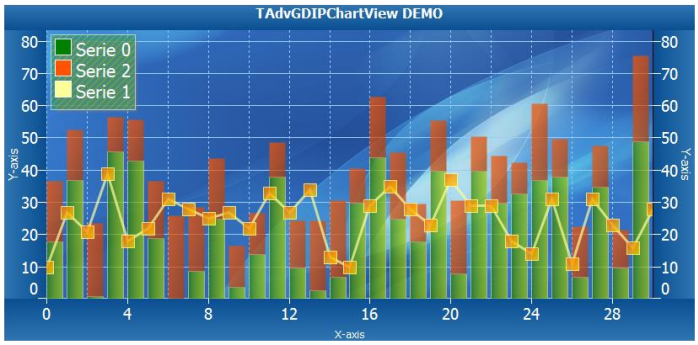
6: ChartType:
Choose a chart type and add markers to mark the Y-value of the point.

7: Annotations:
Add an annotation to add text to important points in the series.

When a New TAdvChartView component is dropped on the form, the component contains one pane with 3 series. Double-click the TAdvChartView component to popup the pane editor that allows adding and removing series from the pane.

TAdvChartViewGDIP (GDI +)

The GDI+ component descends from TAdvChartView and adds many GDI+ features: anti-aliasing, alpha transparency, complex gradients, shadows, textures, hatches …. TAdvChartViewGDIP was specifically designed separately from TAdvChartView as TAdvChartViewGDIP depends on the Microsoft GDIPLUS.DLL library which is available by default on Windows Vista and Windows XP but not on Windows 2000. For Windows 2000, GDIPLUS.DLL can be simply deployed along with the application executable. If you do not want the GDIPLUS dependency, you can use TAdvChartView.

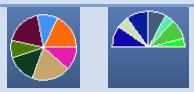
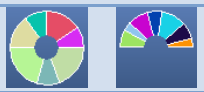
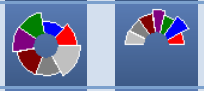
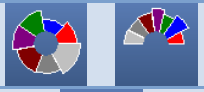


TAdvChartTypeSelector

Choosing the right chart type to visualize your data depends on the data you are using. Below is an overview of the current available chart types and when to use them. If you Double-click on the TAdvChartView component at design time, a pane editor is shown to change properties of background, crosshair, legend, navigator… You can access the Series in the current pane and change the chart type. In code, the equivalent to do this is:

AdvChartView.Panes[PaneIndex].Series[SerieIndex].ChartType := ctLine

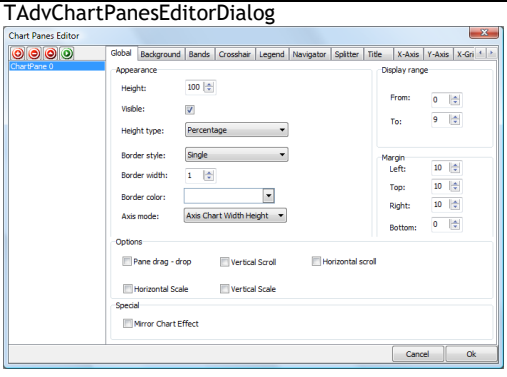
Name	Type	Description
ctNone		Series is not displayed
ctLine		Series is shown as a line from value to value
ctDigitalLine		Series is shown as a digital line from value to value
ctBar		Series is shown as bars with height representing the value. The bar can be a rectangle, pyramid or cylinder.
ctArea		Series is shown as a filled area
ctLineBar		Series is shown as bars with values connected by a line
ctHistogram		Series is shown as a histogram which is similar to a bar chart except that it offers a different color for each value.
ctLineHistogram		Series is shown as a histogram with values connected by a line

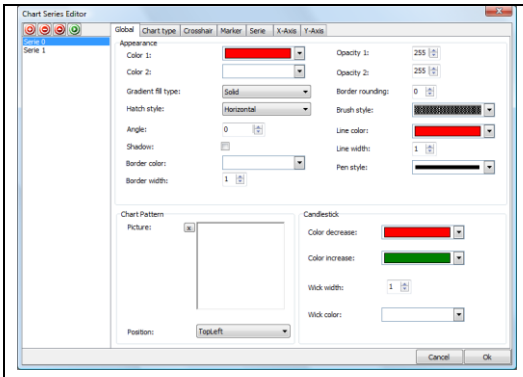
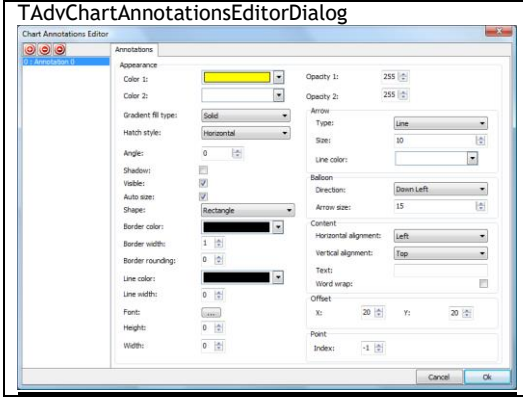
ctCandleStick		Series is shown as a candlestick chart, showing 4 values per point : open/close & high/low value
ctLineCandleStick		Series is shown as a candlestick chart with values connected by a line
ctOHLC		Series is shown as OHLC chart, showing 4 values per point : open/close & high/low value
ctMarkers		Series is shown as marker/image per value
ctStackedBar		Multiple series joined in stacked bar which shows the summed value of all series of type ctStackedBar
ctStackedArea		Multiple series joined in stacked bar which shows the summed value of all series of type ctStackedArea
ctStackedPercentageBar		Same as stacked bar but values per series are represented by percentage
ctStackedPercentageArea		Same as stacked area but values per series are represented by percentage
ctError		Two values per series point show at value with line height depending on error value.
ctArrow / ctScaledArrow		Two values per series point show an arrow. The vertical position of the arrow depends on the first value, the orientation of the arrow depends on the second value
ctBubble / ctScaledBubble		Two values per series point show a circle. The vertical position of the circle depends on the first value, the size of the circle depends on the second value
ctPie/ctHalfPie		Add points and display in an (optionally) exploded pie/halfpie chart with many options like slice gradient colors and values.
ctDonut/ctHalfDonut		Add points and display in an (optionally) exploded donut/halfdonut chart with the possibility of multiple donut rings.
ctSpider/ctHalfSpider		Add points and display in a spider chart with optional grid.
ctBand		Add double points to create a band type chart (similar to area but with without zero points)
ctVarRadiusPie/ctVarRadiusHalfPie		Add points and display in a variable radius pie with optional grid
ctVarRadiusDonut/ctVarRadiusHalfDonut		Add points and display in a variable radius donut with optional grid
ctSizedPie/ctSizedHalfPie		Add points and display in a sized pie with optional grid
ctSizedDonut/ctSizedHalfDonut		Add points and display in a sized donut with optional grid
ctBoxPlot		A Chart type with five-number summaries: the smallest observation (minimum), lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation (maximum).

ctRenko		rising and falling diagonal lines of boxes that are filled and show a downward(red) or upward(green) trend
ctXYLine		Series is shown as a line from value to value, with a custom X value.
ctXYMarkers		Series is shown as marker/image per value, with a custom X value.
ctFunnel		Series is shown as a funnel chart type with 2 modes and optional 3D enabled drawing. The funnel chart has properties that are accessible directly on Serie level and Funnel subproperties. Uses the following properties on Funnel level: - Left - Legend* properties - Position - ShowLegendOnSlice - ShowValues - Top - ValueFont - ValuePosition
Extra information		The difference between variable radius pie and sized pie is that the sized pie angle is equal for all slices (10 slices = 36°). The radius in the variable radius pie is calculated based on the added values.

TAdvChartPanesEditorDialog, TAdvChartSeriesEditorDialog, TAdvChartAnnotationsEditorDialog

All properties of the TAdvChartView component can be edited at runtime via components in the same way as at design-time. Therefore three editors are installed in the component palette to easily change properties of the pane, series and annotation elements. In each separate dialog you can remove existing or add new elements and simply cancel or save your changes.

TAdvChartPanesEditorDialog 	The Panes editor dialog is commonly used to change the appearance of the pane. Properties such as Background, CrossHair, Legend, Navigator, Splitter, Title, X-axis, Y-axis, X-grid and Y-grid are all fully customizable to change the look and feel. Open the Pane Editor Dialog by double-clicking on the TAdvChartView/TAdvChartViewGDIP component.
TAdvChartSeriesEditorDialog	The Series editor dialog is used to change the appearance of the series, and to change the calculation of the series values. An important property Autorange at the Y-axis tab changes the way the minimum and the maximum value

	are calculated. The Series editor is called by double-clicking on the name of the pane in the pane editor or by clicking on the green button in the toolbar.
TAdvChartAnnotationsEditorDialog 	The Annotations editor dialog is a smaller editor which only edits the annotations attached to the series. Annotations are used to mark important points or to display comments about the points. You can choose different types of shapes such as balloon, circle, square… The Annotations editor is called by double-clicking on the name of the series in the series editor or by clicking on the green button in the toolbar.

Introduction to programmatic use of components in TMS VCL Chart

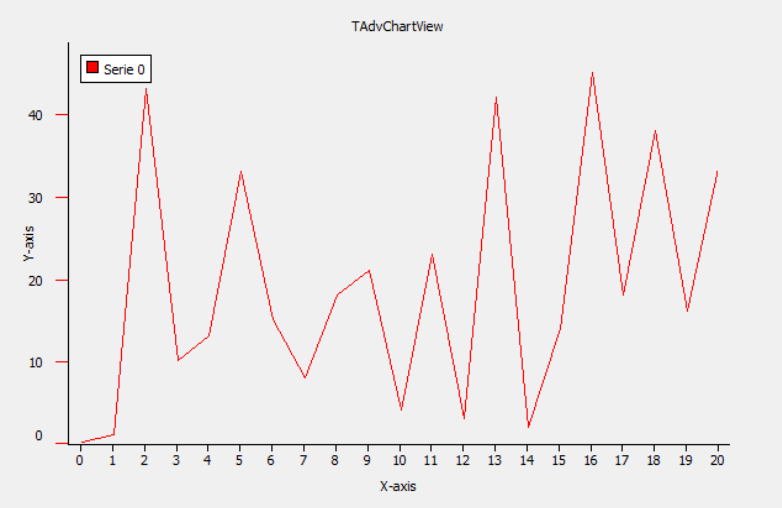
TAdvChartView

To visualize points the TAdvChartView component must be filled with values. Therefore the method AddSinglePoint adds points to a specific chart pane with a specific chart series. After adding the points, set the Range of points you want to have visualized in the chart. In the sample below the range is set from 0 to 20 to display all 20 values. Note: Include the AdvChart Unit to use the value arEnabled for the AutoRange property as this type is defined in the unit AdvChart.

Example:

```
procedure TForm1.AddPoints;
var
  i: integer;
begin
  //Adds 20 Points with random value from 0 to 50 to the first Pane (0)
  //with the first Series (0)
  for i := 0 to 20 do
    AdvChartView.Panes[0].Series[0].AddSinglePoint(RandomRange(0, 50));

  //Set Range from 0 to 20
  AdvChartView.Panes[0].Range.RangeFrom := 0;
  AdvChartView.Panes[0].Range.RangeTo := 20;
  //Set Auto Display Range to arEnabled
  AdvChartView.Panes[0].Series[0].Autorange := arEnabled;
end;
```

Result:



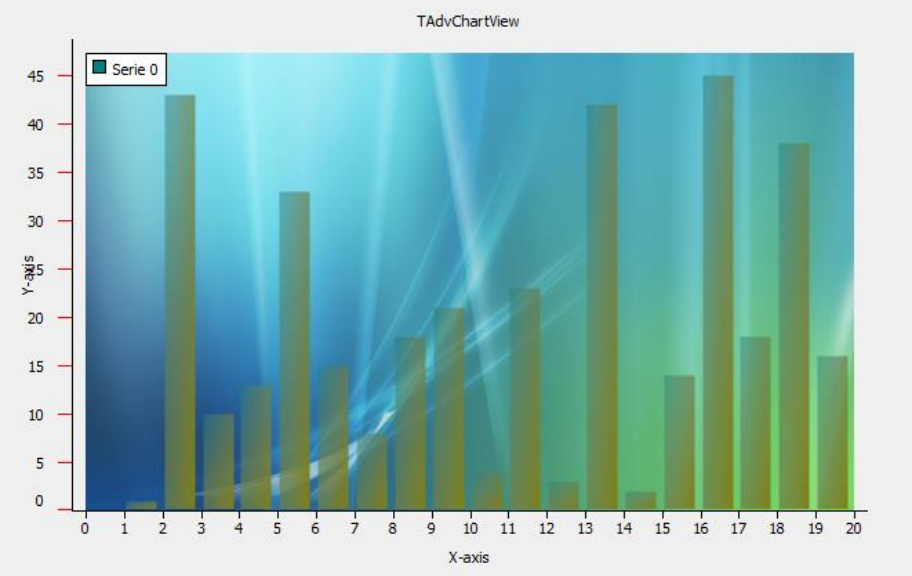
TAdvChartViewGDIP (GDI +)

To visualize points, use the same method for adding points as for the TAdvChartView component. To use transparency and complex gradients add a TAdvChartViewGDIP component to the form and add the code below. Add Unit AdvChartUtil to the Uses clause. The best way to see the transparency is to add a Background image.

Example:

```
procedure TForm1.GDIPGraphics;
begin
  //Adds Gradient start color and Gradient end color with a Forward
  //Diagonal Gradient type. The starting Transparency is 120 and the end
  //transparency is 255;
  AdvGDIPChartView.Panes[0].Series[0].Color := clTeal;
  AdvGDIPChartView.Panes[0].Series[0].ColorTo := clOlive;
  AdvGDIPChartView.Panes[0].Series[0].Opacity := 120;
  AdvGDIPChartView.Panes[0].Series[0].Opacity := 255;
  AdvGDIPChartView.Panes[0].Series[0].GradientType := gtForwardDiagonal;
end;
```

Result:



TAdvChartTypeSelector

The TAdvChartTypeSelector component is used in the Series editor dialog to choose a different chart type. It can also be separately used. The component shows a mini representation of the chart type. You can drop several TAdvChartTypeSelector components on the form, select the chart type for each TAdvChartTypeSelector and add code to the ChartType MouseDown event to set the selector in Selected mode and get the chart type of the chart type selector.

Example:

```
procedure TForm1.ChartTypeSelector1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  (Sender as TAdvChartTypeSelector).Selected := true;
  AdvChartView.Panes[0].Series[0].ChartType := (Sender as
TAdvChartTypeSelector).ChartType;
end;
```

TAdvChartPanesEditorDialog, TAdvChartSeriesEditorDialog, TAdvChartAnnotationsEditorDialog

To use the pane, chart or annotations editors at runtime simply drop the dialog component on the form, set the chart property to the TAdvChartView you want to edit and add the following code:

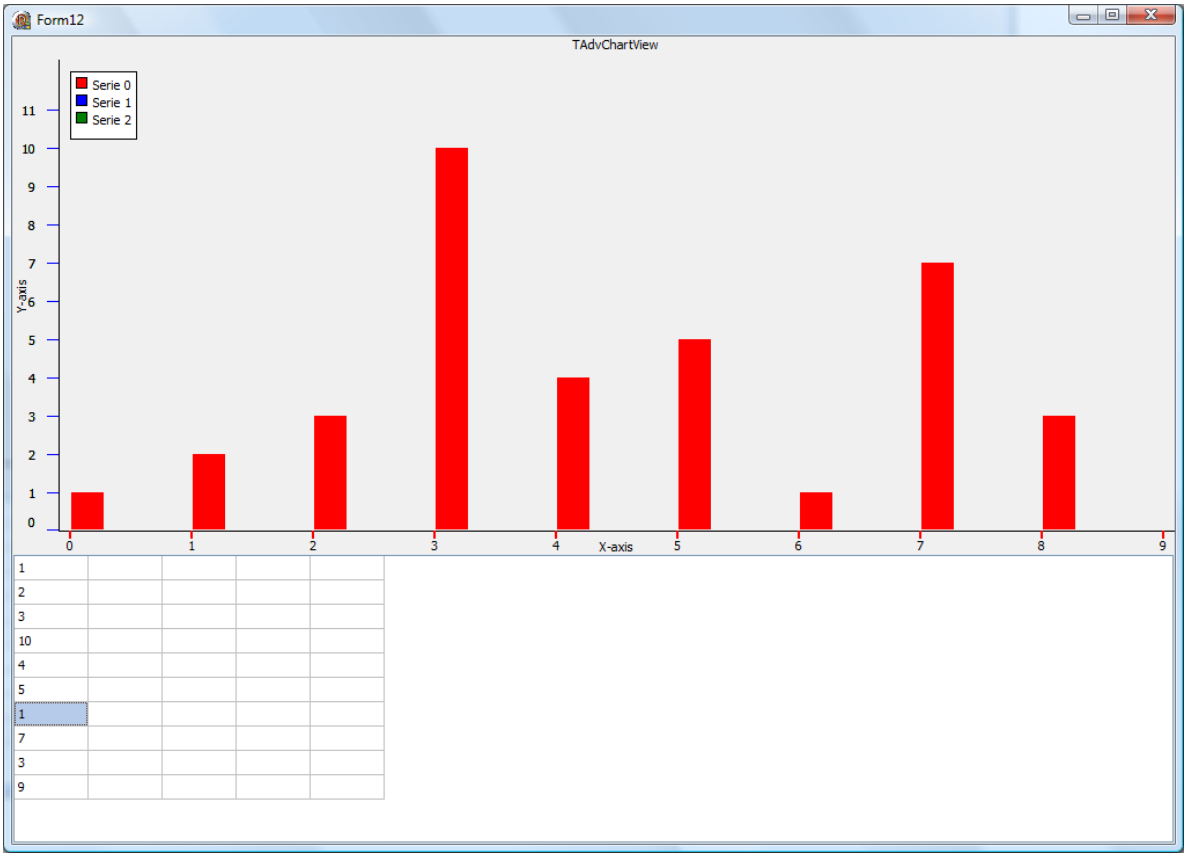
Example:

```
procedure TForm1.LoadPaneEditor;
begin
  AdvChartPanesEditorDialog.ChartView := AdvChartView1;
  AdvChartPanesEditorDialog.Execute;
end;
```

TAdvChartLink

This is a non-visual component and links a TAdvChartView with a TAdvStringGrid component. Please note that TAdvStringGrid is not included in TMS VCL Chart. It is separately available at http://www.tmssoftware.com/site/advgrid2.asp. To start, drop a TAdvChartLink, TAdvChartView, TAdvStringGrid component on the form and set the Chartview and Grid properties in the TAdvChartLink component. To Edit the TAdvStringGrid set the goEditing property true in the Options set. The TAdvChartLink component has several properties.
First the active property must be set true. To tell the TAdvChartLink which column in the grid will be used to display the data you must select an option from the DataType property. For example select dtFullColumn. Then run the project and type your values in the first column of the grid. While typing, the chart will be updated.

Result:



Applying a 3D effect to chart series

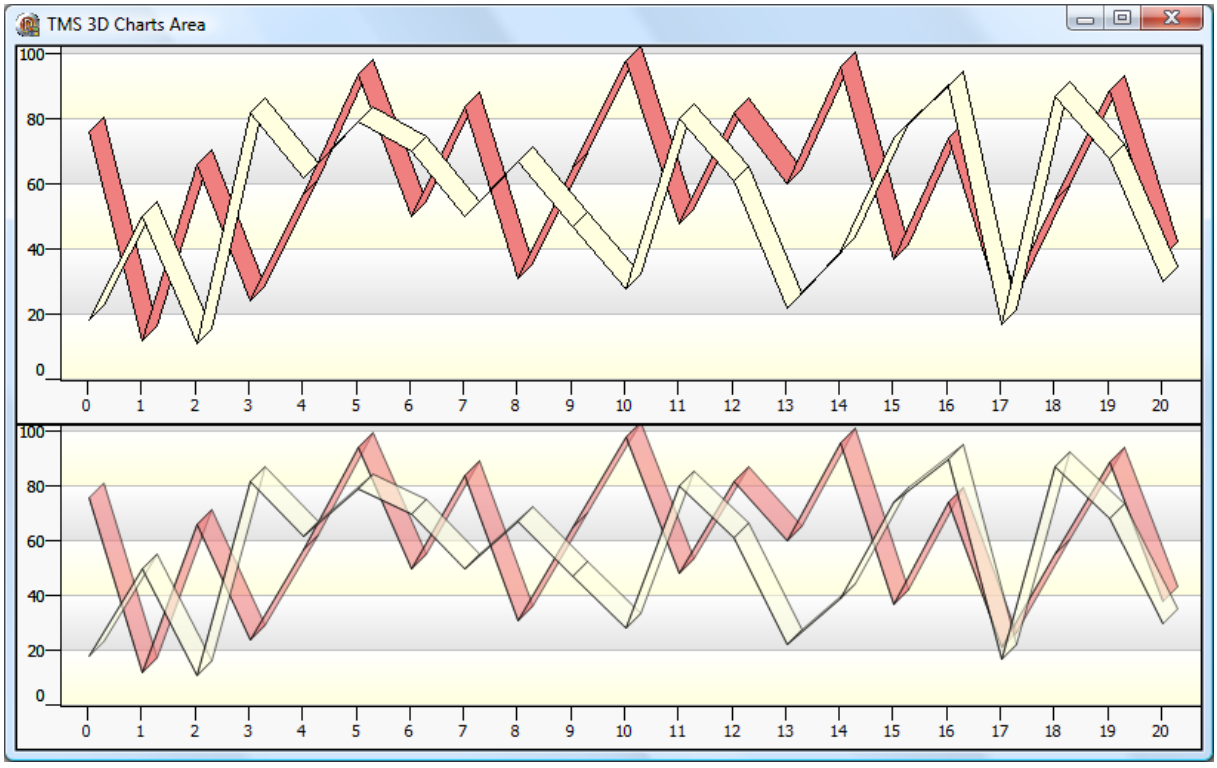
It is possible to apply a "3D effect" on the chart series. This effect can be customized with a 3D Offset property. This is a property that controls the depth of the 3D view. In code, the equivalent to do this is:

```
AdvChartView.Panes[PaneIndex].Series[SerieIndex].3D := true;
AdvChartView.Panes[PaneIndex].Series[SerieIndex].3DOffset := 20;
```

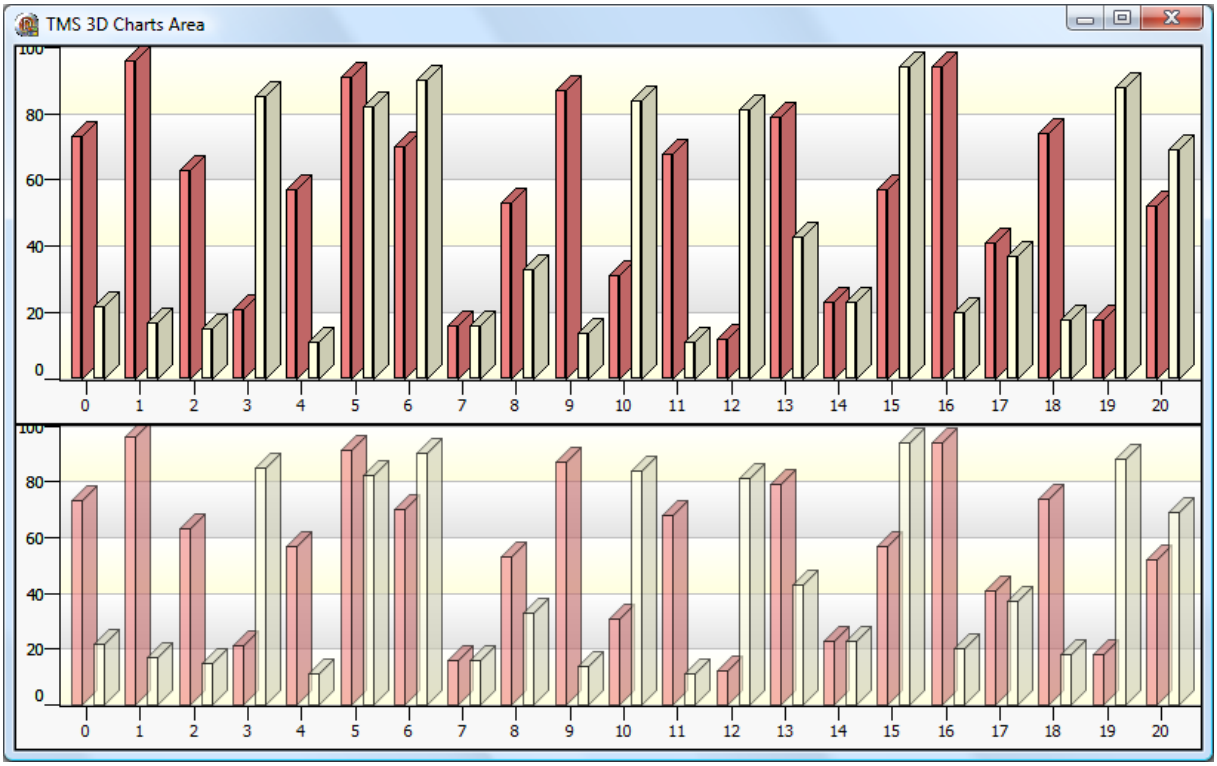
Not all chart types support the 3D view. Below is a list of chart types which implement the 3D effect.

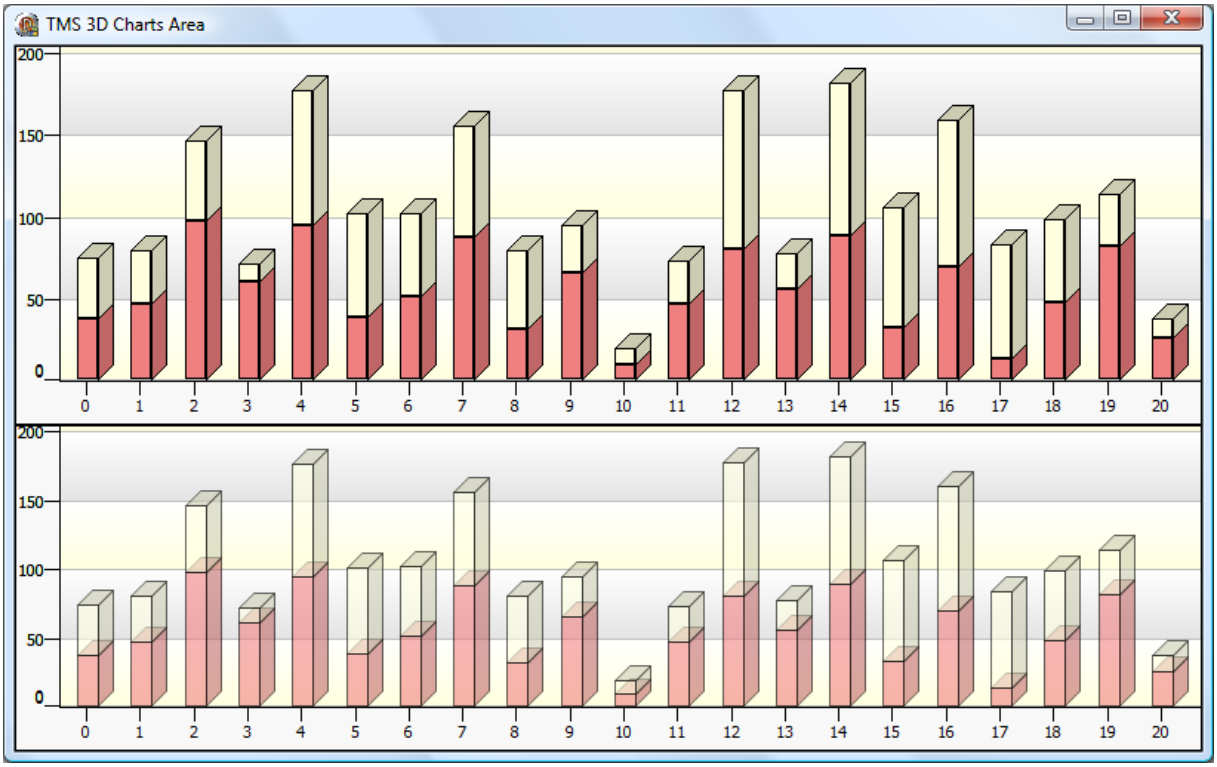
Please note that the above chartview is the TAdvChartView with GDI graphics and the second chart is the TAdvGDIPChartView that is drawn with GDI+ graphics.

Line

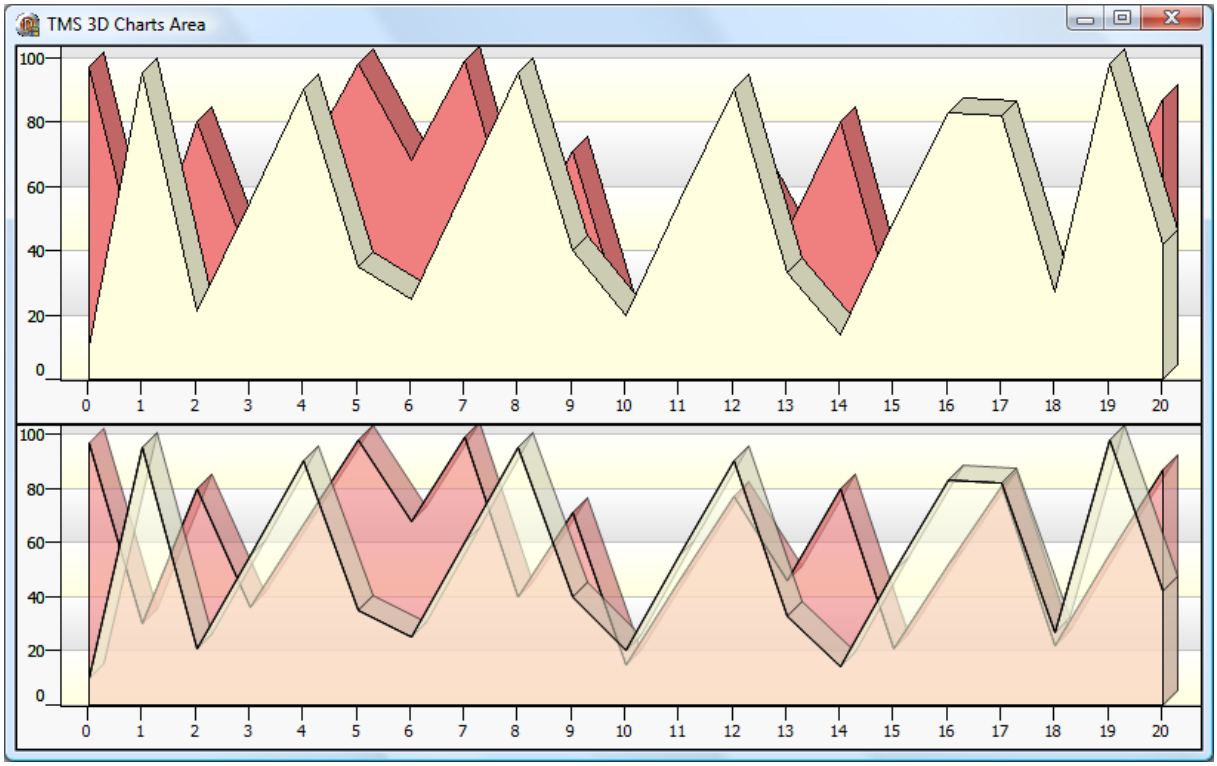


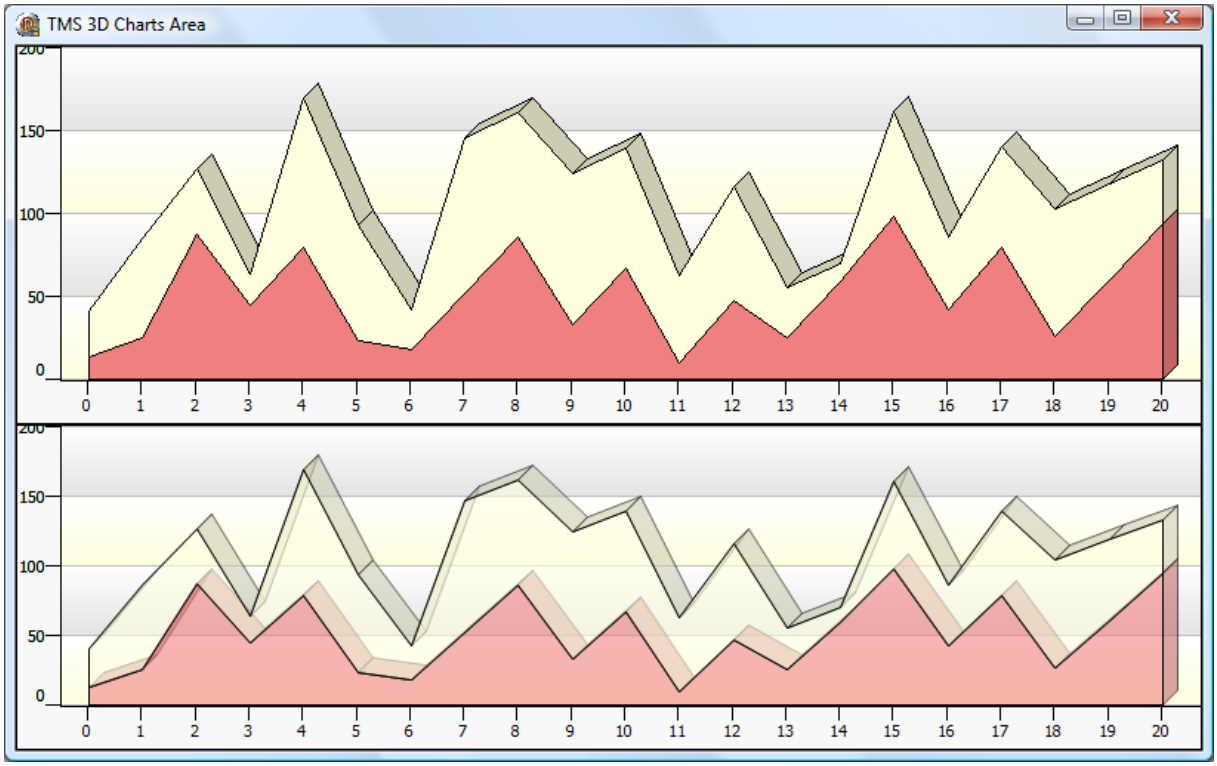
Stacked Bar & Normal Bar



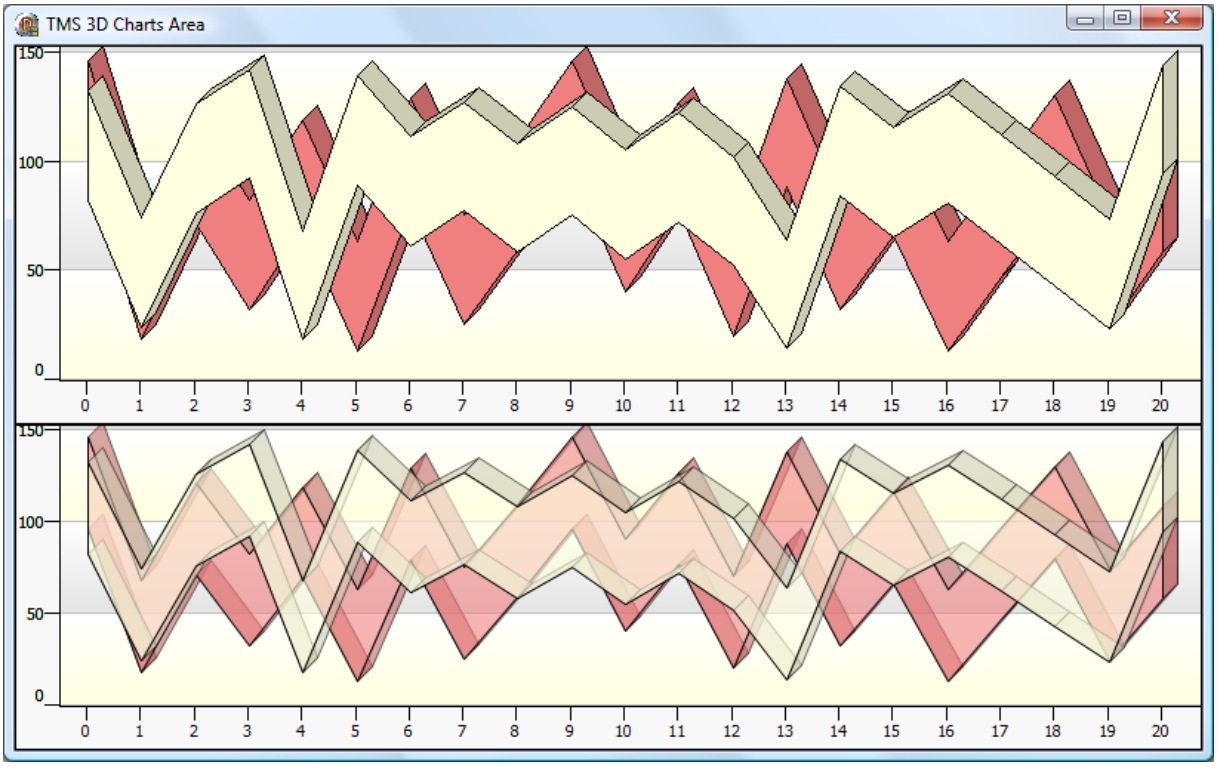


Stacked Area & Normal Area





Band



Note:
ctLineHistogram, ctLineBar, ctStackedPercentagebar, ctStackedPercentageArea also implement the 3D effect.

TMS VCL Chart important methods and properties.

TAdvChartView

Tracker

To track and display values found at crosshair position you can use a floating tracker window. Programmatically, this can be enabled with:

```
AdvChartView1.Tracker.Visible := true;
```

The tracker window is displayed on top of the chart component and can be moved anywhere on the screen. The tracker show the series and the series values found at mouse cursor position on the chart while moving with the mouse cursor.

By default the tracker window is autosized to its content.
With `AdvChartView1.Tracker.AutoSize := false;` you can change the width and height of the tracker.

You can make the Tracker window transparent by setting AlphaBlend property to true and changing the AlphaBlendValue. Several other properties are available like BorderStyle, Color and ColorTo, GradientDirection, Font … to further customize the appearance of the tracker.

TChartPanes

Global

Properties
Programmatically all properties of the Pane at PaneIndex can be accessed with:

Delphi code:

```
AdvChartView1.Panes[PaneIndex]."property"
```

C++Builder:

```
AdvChartView1->Panes->Items[PaneIndex]->"property"
```

Property	Value	Description
AxisMode	amAxisChartWidthHeight amXAxisFullWidth amYAxisFullHeight	The Axismode allows you to change the drawing of the X-axis and Y-axis amAxisChartWidthHeight:  amXAxisFullWidth:  amYAxisFullHeight: 
Height	Integer	Set the height in combination with heighttype to a fixed or autosize value.
HeightType	htPixels htAuto	The type of the height when you have multiple panes.

	htPercentage	**Example** There are 3 panes of height = 30. htPixels: total height will be 90 htAuto: total height will be the AdvChartView component height and the panes height will be auto calculated htPercentage: total height will be 90 % of the AdvChartView component height
Name	String	The name of the pane. In combination with AdvChartView1.GetPaneByName('sample') you can access a pane with a specific name.
Options	poMoving poHorzScroll poVertScroll poHorzScale poVertScale	Set the pane options to allow pane moving, horizontal scrolling, vertical scrolling, horizontal scaling and vertical scaling. Please note that Options is a Set of TPaneOptions and that it is possible to have multiple options enabled.

Methods

Adding and removing a Pane: Open the Pane Editor dialog and click on the "-" icon to remove the selected pane. Click on the "+" icon to add a new pane. Below is a sample code snippet to add and remove a pane:

```
procedure TForm1.AddNewPane;
var
  p: TAdvChartPane;
begin
  p := AdvChartView.Panes.Add;
  // set properties of Pane here
end;
```

In case the GDI+ TAdvGDIPChartView is used, this becomes:

```
procedure TForm1.AddNewPane;
var
  p: TAdvGDIPChartPane;
begin
  p := AdvGDIPChartView.Panes.Add;
  // set properties of GDI+ Pane here
end;
```

To remove a pane, this code can be used:

```
procedure TForm1.RemovePane;
begin
  AdvChartView.Panes[0].Free;
end;
```

Scaling and scrolling: The chart pane can be scaled in both X-direction and Y-direction. Set the property VerticalScale and/or HorizontalScale in the pane options to true. Then click and hold your mouse on the X-axis or Y-axis to scale the chart. You can also scroll left, right, up or down. Set the HorizontalScrolling and/or VerticalScrolling true, click on the pane background and hold your mouse to scroll.

Pane moving: The chart pane can be moved to another position in the Chartview. Set the Pane drag & drop property true in the pane options. Hold Shift + click and drag the pane above another pane. Then release the Shift + click and the pane will be dropped into the new location.

Crosshair

Properties

The crosshair behaviour can be set per pane. Programmatically all properties of the crosshair at PaneIndex can be accessed with:

```
AdvChartView1.Panes[PaneIndex].CrossHair.”property”
```

Properties	Value	Description
CrossHairType	chtNone chtSmallCrossHair chtFullSizeCrossHairAtCursor chtFullSizeCrossHairAtSeries	The Crosshair type you want to use when crosshairs are enabled. chtNone: no crosshair is visible chtSmallCrossHair: a Small crosshair will appear when selecting a point in range between rangefrom and rangeto chtFullSizeCrossHairAtCursor: A full pane width horizontal crosshair will appear at cursor point chtFullSizeCrossHairAtSeries: A full pane width horizontal crosshair will appear at every series.
CrossHairYValues	ShowSerieValues: Boolean ShowYPosValue: Boolean Position: TChartCrossHairYValuePosition (chYAxis chAtCursor chValueTracker)	ShowSerieValues enables you to show the horizontal crosshair found with the vertical crosshair at every series. ShowYPosValue enables you to show the horizontal crosshair found with the vertical crosshair on the Mouse - position

		The Position property allows you to set the values that are found on the intersection of the vertical and horizontal crosshair. You can allow the value to be displayed at the mouse cursor on the horizontal crosshair or at the value tracker window if it is visible.

Methods

Searching values with the Crosshair: There are 3 kinds of crosshair types you can use to locate the Y-values of a series:

1) Small crosshair that shows if you hover with the mouse over a point in the series.

2) Full size crosshair at cursor that follows the mouse movements in vertical and horizontal direction.

3) Full size crosshair at series that only follows the mouse in vertical direction and searches for a point in horizontal direction.

Several other properties in the crosshair class can be used to change the thickness and color of the crosshair line.

It is also possible to programmatically retrieve the value of a series at the crosshair. This code snippet below will put the series values at crosshair in the caption of the form. The code can be added for example to the OnMouseMove event of the chart.

```
var
  i: integer;
  cp: TChartPoint;
  s: string;
begin
  with AdvChartView1.Panes[0] do
  begin
    s := '';
    for i := 0 to Series.Count - 1 do
    begin
      cp := GetChartPointAtCrossHair(i);
      if s = '' then
        s := floattostr(cp.SingleValue)
      else
        s := s + ':' + floattostr(cp.SingleValue);
    end;
  end;
  Caption := s;
end;
```

Navigator

Scaling and scrolling with the navigator: Instead of scrolling on the pane and scaling with the Y-axis and the X-axis you can show a navigator on the pane. With the navigator you can

only scroll and scale in X-direction. To scale, hold your mouse on the navigator and move left or right to zoom in / out. To scroll, set the Scrollbuttons property to visible and click and hold on the left button to scroll left, or the right button to scroll right.

Splitter

Changing pane height with the splitter: To change the pane height, move your mouse cursor until it changes to a crSizeNS type (Sizing direction North – South). Then click and hold your mouse and drag up or down. The splitter is always attached to the pane under the splitter. You can only change the height of the pane if you have multiple panes and the Pane height type is set to pixel or percentage. If you move up with the mouse your pane will increase height and the pane above will decrease height.

X-axis

Properties

The X-axis formatting can be set per pane and per series. Programmatically all properties of the X-axis at PaneIndex can be accessed with:

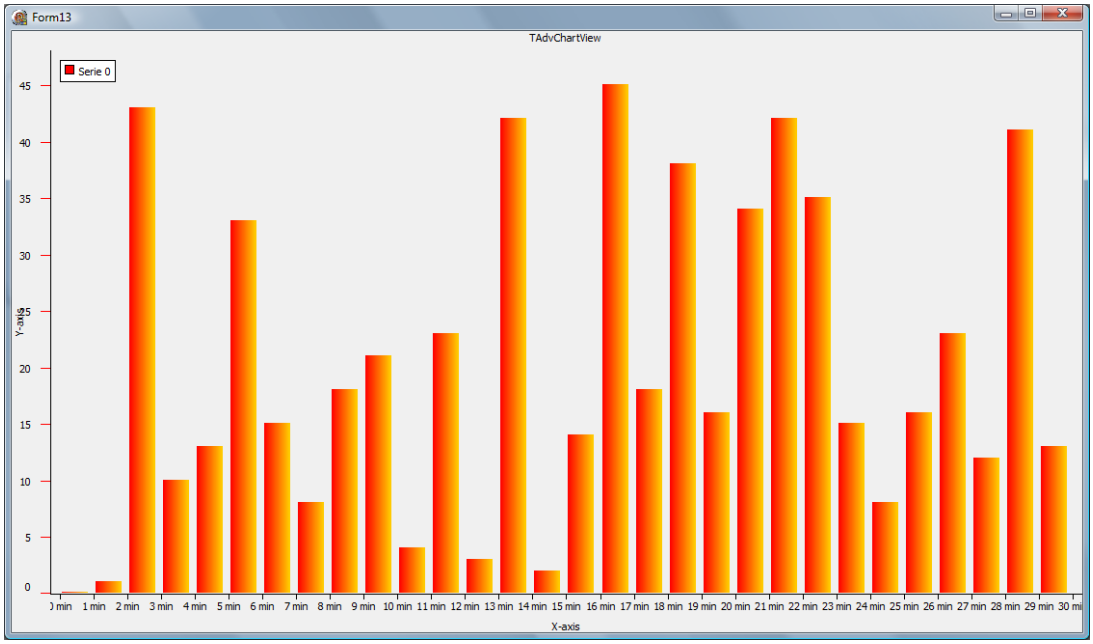
```
AdvChartView1.Panes[0].XAxis.”property”
```

Properties	Value	Description
Position	xBottom xTop xBoth xNone	This property changes the position of the X-axis to top, bottom or will display a X-axis at the top and the bottom in case the position is xBoth
UnitType	utNumber utDay utMonth utYear utHour utMinute	The unittype is utNumber by default and will display values from rangefrom to rangeto In case you choose another value you must add a Timestamp in AddSinglePoint or AddMultiPoints and specify a MajorUnitTimeFormat in the Serie.XAxis
AutoSize	True / False	AutoSize is False by default. If set true the X-Axis size is automatically calculated based on the text and the number of series that have visible X-Axis text

Methods

Setting the UnitType of the X-axis: 2 types for automatic display of the X-axis values are available: the standard number format, the date/time format. Choosing the date/time format, you must specify which date/time unit type needed. By default the number format is set that shows the X-axis as consecutive numbers 0,1,2…
Example: Points have been added to the series that represent values sampled every minute per point. So after 30 minutes you will have 30 points. Change the X-axis UnitType property to utMinute to display the X-axis values per minute. To format the shown values set the Major – Minor time unit format in the Series X-axis property to “ n:’ min’ “.
If it is not desired that auto numbering or auto date/time values are shown, set the value

for the X-axis for each point in a series, an overload of the AddSinglePoint method can be used where the value is set via an extra parameter. For further customization, the ChartSerie.OnAXisDrawValue event can also be implemented with custom drawing.



Y-axis

Properties

The Y-axis formatting can be set per pane and per series. Programmatically all properties of the Y-axis at PaneIndex can be accessed with:

```
AdvChartView1.Panes[0].YAxis."property"
```

Properties	Value	Description
AutoUnits	True/False	The AutoUnits property auto calculates the best possible minor and major unit step to display values on the Y-axis between the Minimum value and Maximum value of the series. In case AutoUnits is set to false a MajorUnit must be set to range between Minimum and Maximum.
Position	yNone yLeft yRight yBoth	This property changes the position of the Y-axis to left, right or will display a Y-axis at the left and the right in case the position is yBoth
AutoSize	True/False	AutoSize is False by default. If set true the X-Axis size is automatically calculated based on the text and the

		number of series that have visible X-Axis text

Methods

Auto calculate Y values with the AutoUnit property: When AutoUnits is set true, the chart calculates the best possible value and position based on the minimum and the maximum value. If AutoUnits is false you need to set the Major and the Minor unit properties to the difference between the first and the second value you want to see. Example there is a range from 0 to 10. Set the major unit to 2 and the minor unit to 1.

Result: MajorUnit ➔ 0 - 2 - 4 - 6 - 8 - 10
MinorUnit ➔ 0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10

X-grid

Properties

The X-grid formatting can be set per pane. Programmatically all properties of the X-grid at PaneIndex can be accessed with:

```
AdvChartView1.Panes[0].XGrid."property"
```

Properties	Value	Description
MajorDistance/MinorDistance	Integer	If the X-grid is visible you can specify a Major or Minor Distance to draw the grid with smaller or larger steps.

Methods

The X-grid line positions are defined in the same way as the X-axis value positions except that there is no AutoUnits property and the MajorUnit and MinorUnit properties must be filled before you can see the X-grid.

Y-grid

Properties

The Y-grid formatting can be set per pane. Programmatically all properties of the Y-grid at PaneIndex can be accessed with:

```
AdvChartView1.Panes[0].YGrid."property"
```

Properties	Value	Description
MajorDistance/MinorDistance	Integer	If the Y-grid is visible you can specify a Major or Minor Distance to draw the grid with smaller or larger steps.
AutoUnits	True/False	If AutoUnits is true the Y-grid will be drawn at the same calculation as the Y-axis values. This means if there is no room to display minor

		units, the Y-grid will disappear. When AutoUnits is false you can use the MajorDistance/MinorDistance on the same way as the X-grid.
SerieIndex	Integer	The Y-grid is related to the series index because each series can have its own Minimum and Maximum. Change the SerieIndex and the Y-grid will draw to match the new series values

Methods

The Y-grid positions are identical to the Y-axis minor and the major unit. The Y-grid also has an AutoUnits property that when set to true forces the chart to calculate the best possible distance between the grid lines.

TChartSeries

Global

Properties

The Series collection can be programmatically accessed with:

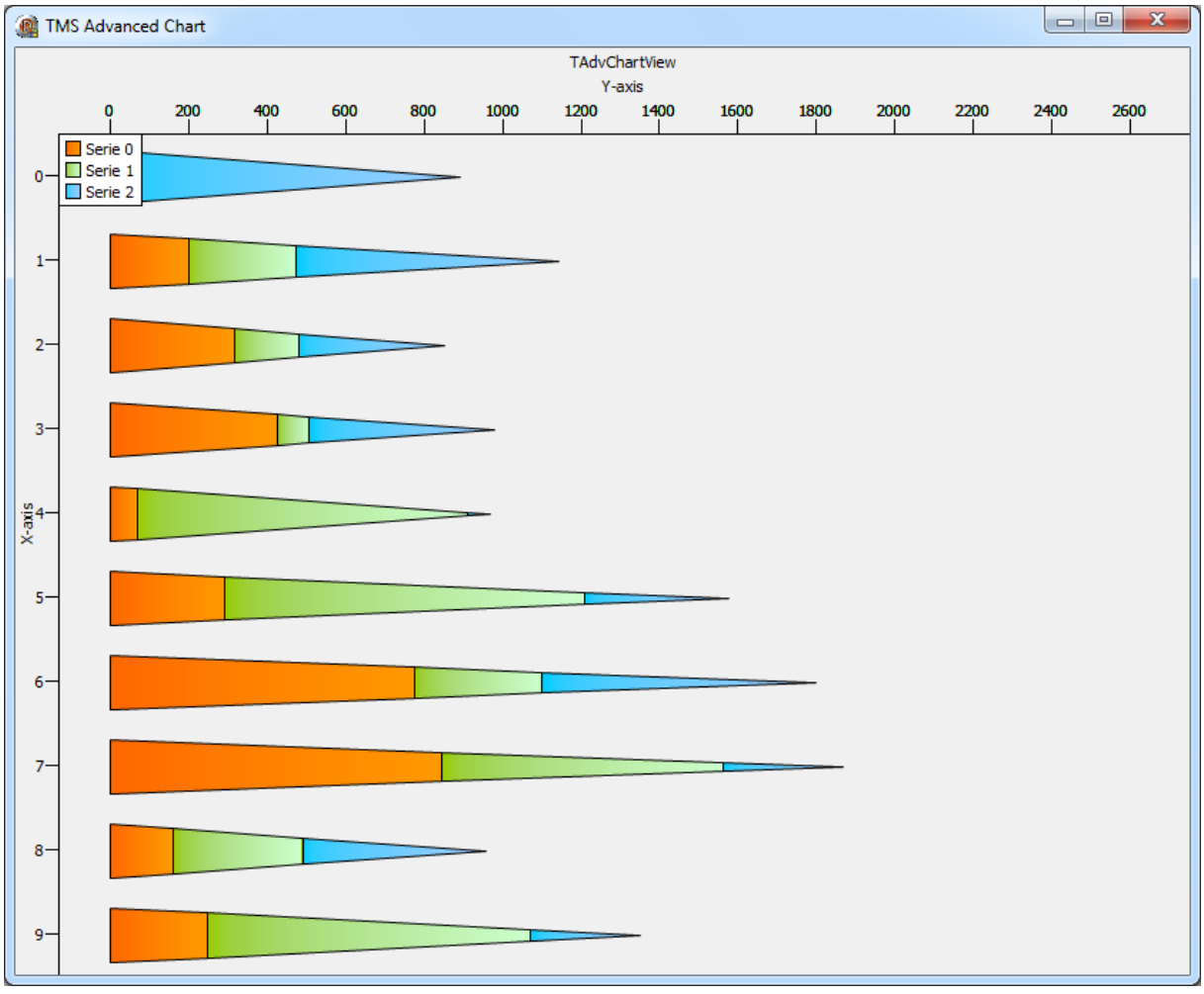
```
AdvChartView1.Panes[0].Series.”property”
```

Properties	Value	Description
ChartMode	dmVertical dmHorizontal	Displays the chart in horizontal or vertical mode This can be applied to every chart type.
BarChartSpacing	Integer	Value between a group of bars
BarChartSpacingType	wtPixels wtPercentage	The type of spacing between a group of bars
DonutMode	dmNormal dmStacked	When using multiple series of the ctDonut charttype, the series can be stacked. Note that this requires the same inner size.
SerieValueTotals	Boolean	When using stacked bars or stacked area’s the values are displayed for each separate series, when SeriesValuesTotal is true the values are summed for each stacked division.

Horizontal Charts

By default, a chart is shown in vertical direction. This means that the X-axis is at the bottom and the Y-axis at the left and/or right side of the chart. In some cases, it is desirable that the chart is shown in horizontal direction. The horizontal mode can be considered as a chart that is rotated -90° degrees. The Y-axis is at the bottom and/or top of the chart and the X-axis on the left side. A single property ChartMode controls the orientation of the chart by setting it to either dmHorizontal or dmVertical.

```
AdvGDIPChartView1.Panes[0].Series.ChartMode := dmHorizontal;
```



TChartSerie

Global

Properties

A series can be programmatically accessed with:

```
AdvChartView1.Panes[0].Series[0]."property"
```

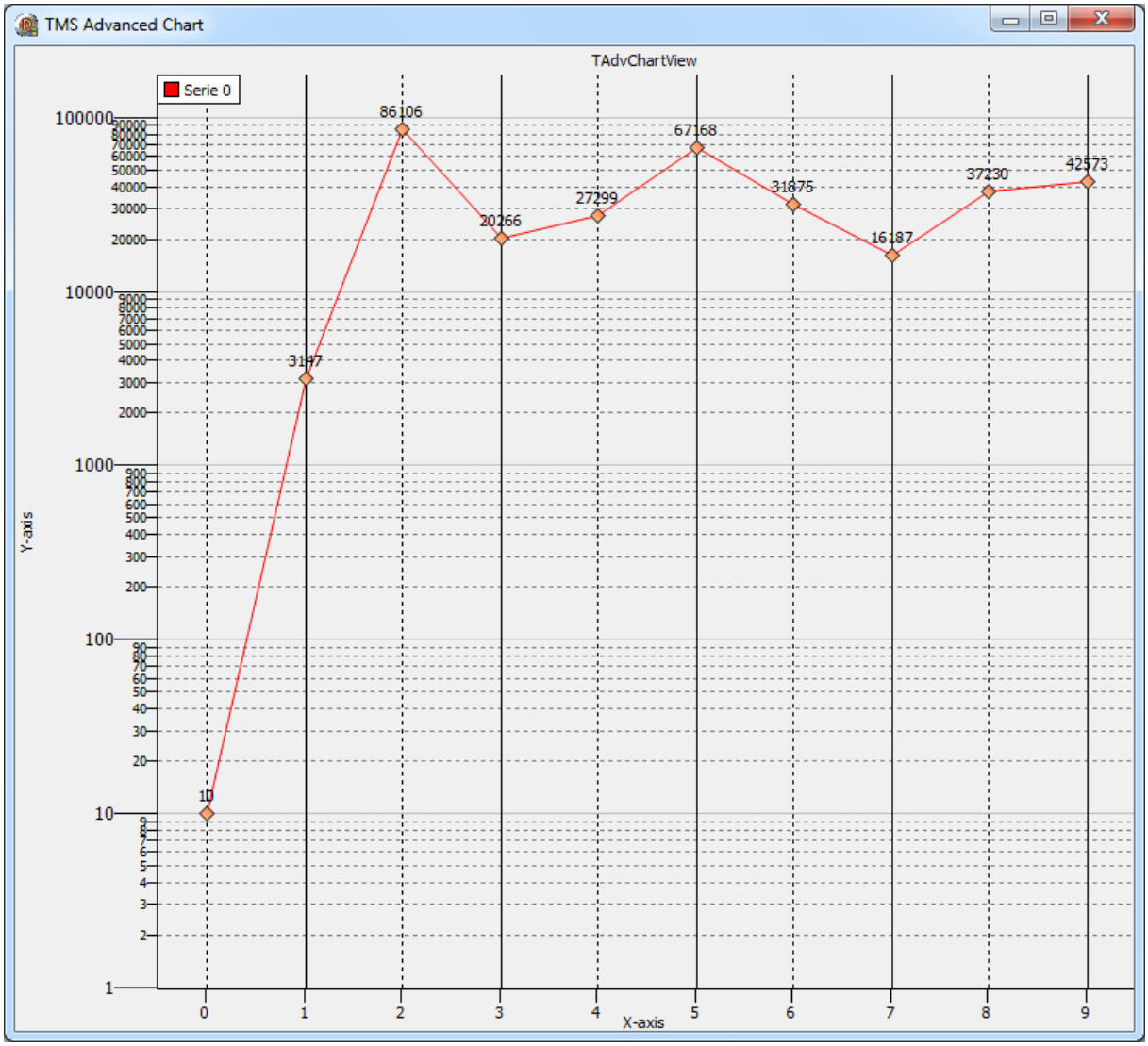
Properties	Value	Description
AutoRange	arDisabled arEnabled arEnabledZeroBased arCommon arCommonZeroBased	The Autorange of the Series to display the Y-axis values. Will be discussed in D) Y-axis configuration
Name	String	The name of the series. In combination with AdvChartView1. GetSerieByName['sample'] You can access a series with a specific name.
LegendText	String	The legend text that will be displayed in the legend.

Points	TChartPointArray	Whenever you want to access or modify an existing point at runtime or design time then you can use this collection of TChartPoint records and modify the value.
		
		Example
		
		`AdvChartView1.BeginUpdate;`
		`AdvChartView1.Panes[0].Series[0].Points[0].SingleValue := Random(100);`
		`AdvChartView1.EndUpdate;`
		
		Please note that you must call BeginUpdate and EndUpdate to update the chart at runtime.

Logarithmic	Boolean	Transforms the Linear Y-Scale into a Logarithmic Y-Scale.
BarValueTextFont	TFont	The font that applies to the text on the bars.
BarValueTextAlignment	TAlignment	The alignment that applies to the text on the bars.
BarValueTextType	btCustomValue btXAxisValue	Allows text to be displayed on the bars. In case btCustomValue, a value can be added with an extra event that must be assigned to the series. In case btXAxisValue, the value that is passed trough the AddSinglePoints.

Logarithmic Y-scale

By default, the Y-axis scale on charts is a linear scale. For scientific applications, it is often mandatory to use a logarithmic scale. When using a logarithmic Y-scale the MajorUnit is automatically set to 1 and cannot be changed. This means that on the Y-axis values 1,10,100,1000,… will be shown. The MinorUnit can be set for the Y-Axis and the Y-Grid to a value of choice. Note that the logarithmic Y-scale is only supported for single value chart types. Below is a sample with a line chart and markers.



Methods

YToValue and ValueToY

If you want to know the value at an Y-Position or the Y-Position of a value then these 2 methods will help you access the corresponding value.

```
var
  r: TRect;
  v: Double;
begin
  r := AdvChartView1.Panes[0].Series.SeriesRectangle;
```

```
  v :=
AdvChartView1.Panes[0].Series[0].YToValue(ScreenToClient(Mouse.Cursor
Pos).Y, r);
end;
```

Selecting a marker: When Serie[index].SelectedMark is set true, it is possible to select a point on the series with a mouse click. The selected point can be get or set via the public Serie.SelectedIndex property. Further characteristics of the selection markers on points of the series can be set via Serie[index].SelectedMarkColor, Serie[index].SelectedMarkBorderColor, Serie[index].SelectedMarkSize.

Adding and removing series: Start the SerieEditorDialog and click on the "-" icon to remove or the "+" icon to add a series. You can add multiple series per pane. Here is an example to remove a series and to add a series to the first pane.

```
procedure TForm1.AddNewSerie;
var
  s: TChartSerie;
begin
  s := AdvChartView.Panes[0].Series.Add;
  // set properties of series s here
end;
```

or in case of the GDI+ version:

```
procedure TForm1.AddNewSerie;
var
  s: TAdvGDIPChartSerie;
begin
  s := AdvGDIPChartView.Panes[0].Series.Add;
  // set properties of GDI+ series s here
end;
```

To remove a series:

```
procedure TForm1.RemoveSerie;
begin
  AdvChartView.Panes[0].Series[0].Free;
end;
```

Adding single, multi points with extra parameters: Depending on the chosen chart type (see different chart types) different methods are available to add series data point values to a series. This is done via a number of overload functions of AddSinglePoint and AddMultiPoint. For some chart types, a single value per series data point is sufficient. Other chart types such as a band chart or OHLC chart require 2 or 4 values per series data point. Additional settings per data point might be necessary such as a specific chart X-axis value for a data point or a color of a data point for a histogram.

The easiest way to add points is with the method AddSinglePoint or AddMultiPoint. Alternatively the methods SetSinglePoint or SetMultiPoint can also be used to change values of some data points in a series. Note though that when no data points are added in the series, it is required to set the number of points first with the method SetArraySize() before using SetSinglePoint or SetMultiPoint. In this example we use the Add Methods:

Example: Adding some random points for different chart types.

```
procedure TForm1.AddRandomSinglePoints;
var
  i: integer;
begin
  with AdvChartView.Panes[0].Series[0] do
  begin
    //Applies to ctLine, ctDigitalLine, ctBar, ctArea, ctHistogram,
    ctLineBar,     //ctLineHistogram, ctStackedBar, ctStackedArea,
    ctStackedPercBar,            //ctStackedPercArea, ctMarkers
     for i := 0 to 10 do
       AddSinglePoint(Random(50));

   //Applies to ctArrow, ctError, ctScaledArrow, ctBubble,
   //ctScaledBubble (Random points with X an Y offset)
    for i := 0 to 10 do
       AddPoints(Random(50), XOffset, YOffset);
  end;
end;

procedure TForm1.AddRandomMultiPoints;
var
  i: integer;
begin
  with AdvChartView.Panes[0].Series[0] do
  begin
    //Randomly add some points ranging from 0 to 10
    //Applies to ctCandleStick, ctOHLC, ctLineCandleStick
     for i := 0 to 10 do
       AddMultiPoints(Random(50), Random(50), Random(50), Random(50));

  end;
end;
```

Note: Always set the RangeFrom and the RangeTo to the range of points you have added to the series that you want to visualize..

Example: Adding points for every month of the year.

You can quickly add custom text to the X-axis by using one of the overloaded AddSinglePoint methods.

```
for I := 0 to 11 do
begin
   AdvChartView1.Panes[0].Series[0].AddSinglePoint(Random(100) + 10,
   ShortMonthNames[I + 1]);
end;
```

Various method overloads: When a points at a specific time are added a timestamp must be added as well. If some points have a special color, a color can be added; when some points should be hidden for some reason you can set the 'defined' property, when a custom X-axis text must be drawn, etc…. Here is a list of all methods you can use to add points.

```
procedure SetSingleDateTime(SingleDateTime: TDateTime; index:
integer);

procedure SetSinglePoint(SinglePoint: double; index: integer);
```

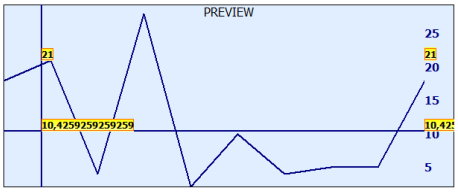
```
procedure AddSingleXYPoint(SingleXPoint, SingleYPoint: double);
overload; (overload for each AddSinglePoint for adding charts with
custom X-Values)

procedure AddSinglePoint(SinglePoint: double); overload;

procedure AddSinglePoint(SinglePoint: double; Defined: Boolean);
overload;

procedure AddSinglePoint(SinglePoint: double; SingleDateTime:
TDateTime); overload;

procedure AddSinglePoint(SinglePoint: double; SingleDateTime:
TDateTime; Defined: Boolean); overload;

procedure AddSinglePoint(SinglePoint: double; Color: TColor);
overload;

procedure AddSinglePoint(SinglePoint: double; Color: TColor; Defined:
Boolean); overload;

procedure AddSinglePoint(SinglePoint: double; SingleDateTime:
TDateTime; Color: TColor); overload;

procedure AddSinglePoint(SinglePoint: double; SingleDateTime:
TDateTime; Color: TColor; Defined: Boolean); overload;

procedure AddSinglePoint(SinglePoint: double; XAxisValue: String);
overload;

procedure AddSinglePoint(SinglePoint: double; Defined: Boolean;
XAxisValue: String); overload;

procedure AddSinglePoint(SinglePoint: double; SingleDateTime:
TDateTime; XAxisValue: String); overload;

procedure AddSinglePoint(SinglePoint: double; SingleDateTime:
TDateTime; Defined: Boolean; XAxisValue: String); overload;

procedure AddSinglePoint(SinglePoint: double; Color: TColor;
XAxisValue: String); overload;

procedure AddSinglePoint(SinglePoint: double; Color: TColor; Defined:
Boolean; XAxisValue: String); overload;

procedure AddSinglePoint(SinglePoint: double; SingleDateTime:
TDateTime; Color: TColor; XAxisValue: String); overload;

procedure AddSinglePoint(SinglePoint: double; SingleDateTime:
TDateTime; Color: TColor; Defined: Boolean; XAxisValue: String);
overload;

(AddSinglePoint also has an equivalent AddDoublePoint to add a second
point in case chart type ctBands is used).

procedure SetMultiPoints(HighPoint, LowPoint, OpenPoint, ClosePoint:
Double; index: integer);
```

```
procedure AddPoints(SinglePoint: Double; PixelValue: integer);
overload;

procedure AddPoints(SinglePoint: Double; PixelValue1, PixelValue2:
integer); overload;

procedure AddPoints(SinglePoint: Double; PixelValue1, PixelValue2:
integer; Defined: Boolean); overload;

procedure AddPoints(SinglePoint: Double; PixelValue1, PixelValue2:
integer; SingleDateTime: TDateTime); overload;

procedure AddPoints(SinglePoint: Double; PixelValue1, PixelValue2:
integer; SingleDateTime: TDateTime; Defined: Boolean); overload;

procedure AddMultiPoints(HighPoint, LowPoint, OpenPoint, ClosePoint:
Double); overload;

procedure AddMultiPoints(HighPoint, LowPoint, OpenPoint, ClosePoint:
Double; Defined: Boolean); overload;

procedure AddMultiPoints(HighPoint, LowPoint, OpenPoint, ClosePoint:
Double; SingleDateTime: TDateTime); overload;

procedure AddMultiPoints(HighPoint, LowPoint, OpenPoint, ClosePoint:
Double; SingleDateTime: TDateTime; Defined: Boolean); overload;

procedure AddMultiPoints(HighPoint, LowPoint, OpenPoint, ClosePoint:
Double; Color: TColor); overload;

procedure AddMultiPoints(HighPoint, LowPoint, OpenPoint, ClosePoint:
Double; Color: TColor; Defined: Boolean); overload;

procedure AddMultiPoints(HighPoint, LowPoint, OpenPoint, ClosePoint:
Double; SingleDateTime: TDateTime; Color: TColor); overload;

procedure AddMultiPoints(HighPoint, LowPoint, OpenPoint, ClosePoint:
Double; SingleDateTime: TDateTime; Color: TColor; Defined: Boolean);
overload;

procedure AddPiePoints(SinglePoint: Double; legendvalue: String = '';
Color: TColor = clNone; ColorTo: TColor = clNone; PieIndent: integer
= 0; Defined: Boolean = true);

procedure AddLineSerie(startx, endx: Integer; startvalue, endvalue:
Double; UseCommonYRange: Boolean = true; SerieLinecolor: TColor =
clBlack; SerieLineWidth: integer = 1);

(AddLineSerie is used to quickly add a line from a start to an end
point).
```

Crosshair

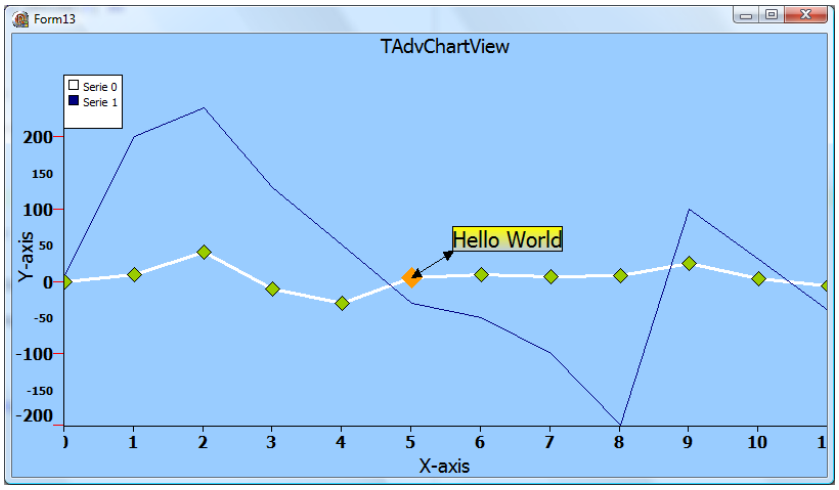
When the crosshair is enabled you can set different graphical settings to display series values or y-position values found at the crosshair. Below is an example crosshair setup at design time by the series editor. With the different types of crosshairs, you can draw horizontal lines where the value was found. The picture below is the crosshair type “full

size crosshair at cursor" containing no horizontal lines except the one where the mouse cursor is. When "full size crosshair at series" crosshair type is chosen a horizontal line will automatically be drawn where a series value is found.



Marker

A marker can be selected. Set the SelectedColor and the SelectedSize property. Click on the marker at runtime, the marker will be shown in the selected color & size. To add code when a marker is clicked you can use the SerieMouseDown or SerieMouseUp event. This can be used for example to add an annotation at a specific point of the series.



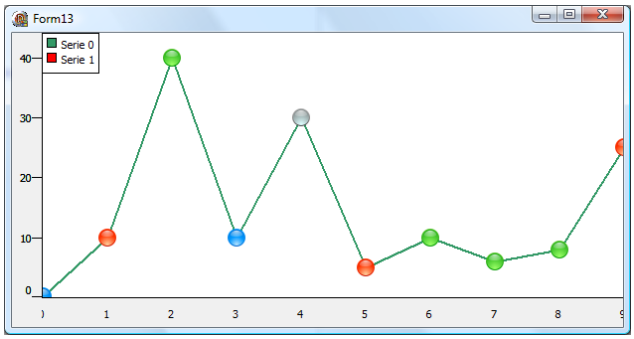
It is possible to customize the drawing of the markers. Select mCustom as marker type and use the event OnMarkerDrawValue. Follow sample code snippet shows how this can be done:

```
type
  TForm1 = class(TForm)
  private
    { private declarations }
  public
    { public declarations }
    procedure DrawMarker(Sender: TObject; Serie: TChartSerie; Canvas:
TCanvas; X, Y,Point: integer; value: TChartPoint);
  end;
```

Assign this procedure to the series for which markers you want to customize the drawing.

```
AdvChartview.Panes[0].Series[0].OnMarkerDrawValue := DrawMarker;
```

Markers also support picture drawing. Select mPicture and load a picture in the markerpicture property. The picture below is an example of the combination of custom and picture drawing.



Y-axis range configuration

Properties

The Y-axis per series can be programmatically accessed with:

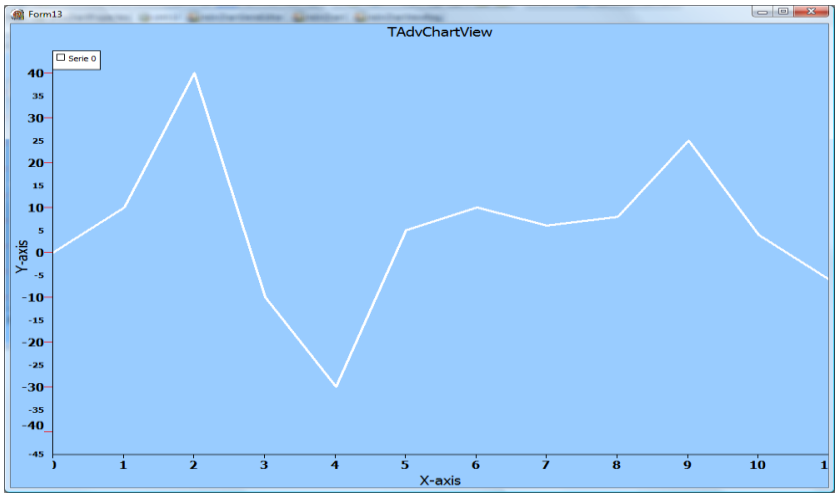
```
AdvChartView1.Panes[0].Series[0].YAxis.”property”
```

Properties	Value	Description
MajorUnit / MinorUnit	Double	When AutoUnits is false on pane level you must specify a MajorUnit and MinorUnit to determine the Y-axis values on Series level

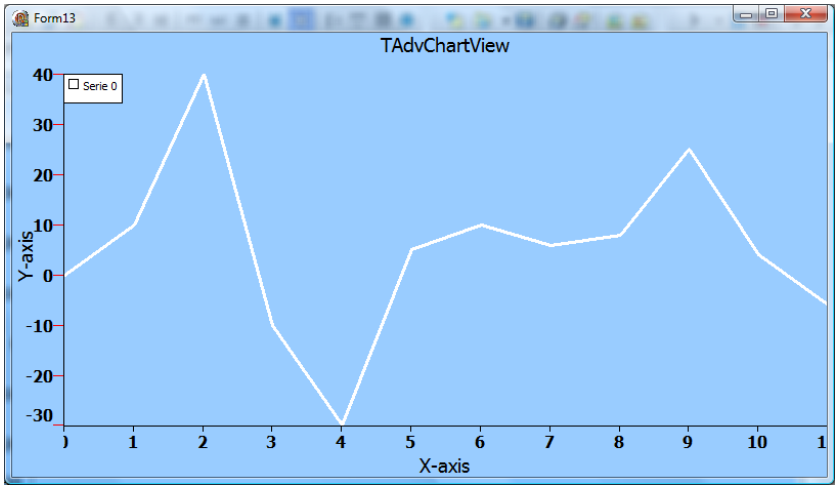
Methods
AutoRange property on the Y-axis: The AutoRange property determines how the series are displayed with an automatically calculated minimum and a maximum value range or with a minimum or maximum that can be programmatically set. There are 5 types you can choose from depending on the chart type you choose. Below is an example which illustrates how the Autorange calculates the Y-axis display range.

Example: Added points: 0, 10, 40, -10, -30, 5, 10, 6, 8, 25, 4, -6

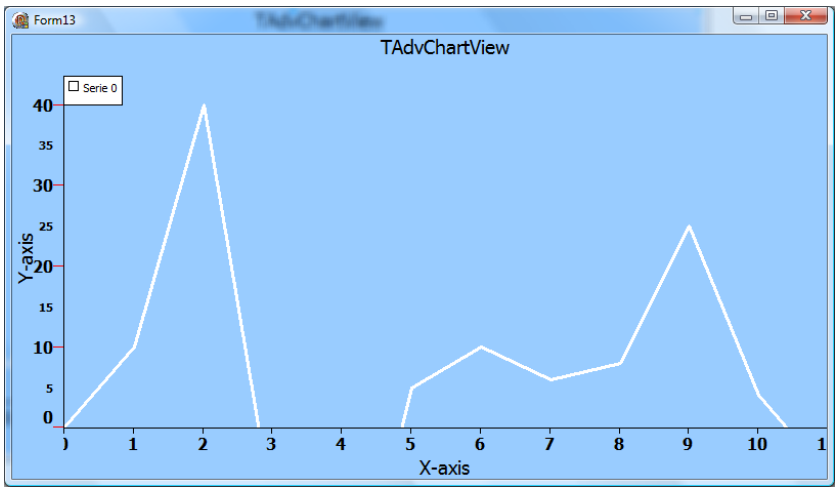
Result 1: Chosen type: arDisabled
If arDisabled is chosen the Chartview does not perform any automatic calculations of minimum or maximum values of the series. If AutoRange is set to arDisabled, the minimum and maximum value must be programmatically set. The minimum or maximum can or cannot correspond to the real minimum and maximum in the series. In case you want to display the series within its real minimum/maximum range, set in this example the minimum value to -30 and the maximum value to 40. This has the same effect as setting the property to arEnabled which calculates the maximum and the minimum of the added points. In the picture below the maximum is set to 45 and the minimum is set to – 45.



Result 2: Chosen type: arEnabled
If arEnabled is chosen the minimum and maximum value are calculated automatically. In the example the minimum value is -30 and the maximum value is 40.

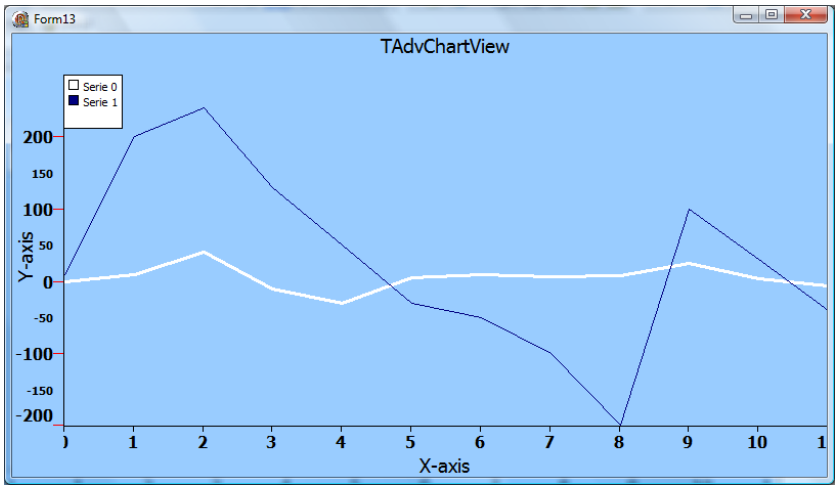


Result 3: Chosen type: arEnabledZeroBased
Again the minimum and maximum values are calculated automatically but this time the minimum value is set to 0 causing all negative values to disappear.

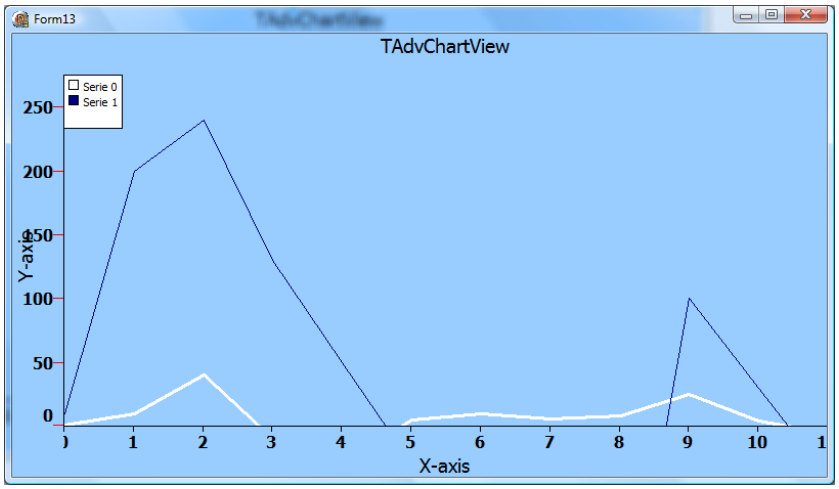


Result 4: Chosen type: arCommon
The arCommon type is only useful if you have more than one series. If you only have one series, the arCommon type will be equivalent as the arEnabled type. The arCommon type can be used to automatically use the minimum/maximum of all calculated minima/maxima of all series with the arCommon type.

Example: We add another series with the values: 10,200,240,130,50,-30,-50,-100, -200, 100,30,-40. The values of the second series are higher in maximum and lower in minimum. If the arCommon type is chosen for both series you will get the result below:



Result 5: Chosen type: arCommonZeroBased
The arCommonZeroBased Autorange type is similar to the arCommon type. The maximum value will still be calculated for multiple series but the major difference with the previous arCommon type is that the minimum value for all series will be set to zero:



Y-axis visual configuration of multiple series values

Different series values on the left and/or right size of the Y-axis: You can add multiple series per pane. Usually the series have a common value range and the Autorange mode arCommon will be chosen. When adding series which do not have a common value range, use the Autorange mode arEnabled, and set property series Y-axis position to left / right / both. You can choose a matching font color for values in the Y-axis that match the series

color for example to easily identify which values belong to which series. Another feature is formatting of the value. Set a format in the Value format property of the series.

Format specifiers have the following form:

"%" [index ":"] ["-"] [width] ["." prec] type

A format specifier begins with a % character. After the % come the following, in this order:

- An optional argument zero-offset index specifier (that is, the first item has index 0), [index ":"]
- An optional left justification indicator, ["-"]
- An optional width specifier, [width]
- An optional precision specifier, ["." prec]
- The conversion type character, type

The following table summarizes the possible values for type:

d: Decimal. The argument must be an integer value. The value is converted to a string of decimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has less digits, the resulting string is left-padded with zeros.

u: Unsigned decimal. Similar to 'd' but no sign is output.

e: Scientific. The argument must be a floating-point value. The value is converted to a string of the form "-d.ddd...E+ddd". The resulting string starts with a minus sign if the number is negative. One digit always precedes the decimal point.The total number of digits in the resulting string (including the one before the decimal point) is given by the precision specifier in the format string—a default precision of 15 is assumed if no precision specifier is present. The "E" exponent character in the resulting string is always followed by a plus or minus sign and at least three digits.

f: Fixed. The argument must be a floating-point value. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative.The number of digits after the decimal point is given by the precision specifier in the format string—a default of 2 decimal digits is assumed if no precision specifier is present.

g: General. The argument must be a floating-point value. The value is converted to the shortest possible decimal string using fixed or scientific format. The number of significant digits in the resulting string is given by the precision specifier in the format string—a default precision of 15 is assumed if no precision specifier is present.Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format.

n: Number. The argument must be a floating-point value. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The "n" format corresponds to the "f" format, except that the resulting string contains thousand separators.

m: Money. The argument must be a floating-point value. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, DecimalSeparator, and CurrencyDecimals global variables or their equivalent in a TFormatSettings data structure. If

the format string contains a precision specifier, it overrides the value given by the CurrencyDecimals global variable or its TFormatSettings equivalent.

p: Pointer. The argument must be a pointer value. The value is converted to an 8 character string that represents the pointers value in hexadecimal.

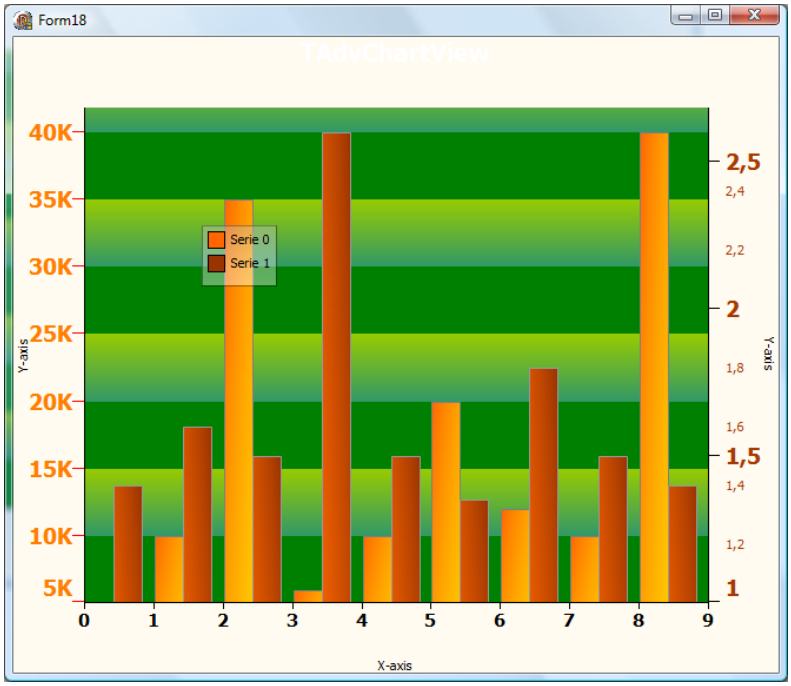
s: String. The argument must be a character, a string, or a PChar value. The string or character is inserted in place of the format specifier. The precision specifier, if present in the format string, specifies the maximum length of the resulting string. If the argument is a string that is longer than this maximum, the string is truncated.

x: Hexadecimal. The argument must be an integer value. The value is converted to a string of hexadecimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has fewer digits, the resulting string is left-padded with zeros.

Conversion characters may be specified in uppercase as well as in lowercase—both produce the same results.
For all floating-point formats, the actual characters used as decimal and thousand separators are obtained from the DecimalSeparator and ThousandSeparator global variables or their TFormatSettings equivalent.

In the sample below, the Y-axis Value format on the left side was set to '%gK' and for the right side '%.1f'

Result:



X-axis

Properties

The X-axis per series can be programmatically accessed with:

```
AdvChartView1.Panes[0].Series[0].XAxis."property"
```

Properties	Value	Description
MajorUnit / MinorUnit	Double	MajorUnit and MinorUnit is used to determine the X-axis values on Series level. If you set the MajorUnit to 2, the values will range from RangeFrom to RangeTo in steps of 2 (0, 2, 4, 6, …)

Methods

Custom drawing on the X-axis: It is also possible to customize the X-axis values drawing. Use the event OnXAxisDrawValue on a series in the same way as the customized marker drawing for example:

```
type
  TForm1 = class(TForm)
  private
    { private declarations }
  public
    { public declarations }
    procedure DrawXAxisValue(Sender: TObject; Serie: TChartSerie;
Canvas: TCanvas; ARect: TRect; ValueIndex, XMarker: integer; Top:
Boolean; var defaultdraw: Boolean);
  end;
```

Then assign this procedure to the series you want to customize the X-axis value drawing.

```
AdvChartview.Panes[0].Series[0].OnXAxisDrawValue := DrawXAxisValue;
```

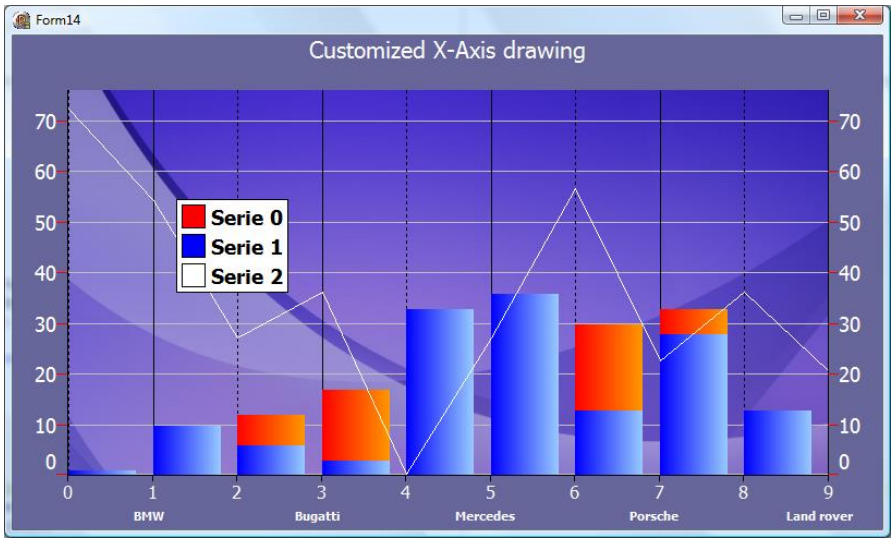
When you want to display the custom drawn value, set defaultdraw parameter to false. Ie. to replace the standard value drawn by TAdvChartView, add code to draw the new value and then set defaultdraw to false, otherwise both values will be drawn on the X-axis.

Example: Add some text on every odd value.

```
procedure TForm.XAxisDrawValue(Sender: TObject; Serie: TChartSerie;
  Canvas: TCanvas; ARect: TRect; ValueIndex, XMarker: integer; Top:
Boolean;
  var defaultdraw: Boolean);

const
 lst: array[0..4] of String = ('Audi', 'BMW', 'Mercedes', 'Bugatti',
'Porsche');
var
  s: String;
  th, tw: integer;

begin
  if Odd(ValueIndex) then
  begin
    Canvas.Font.Size := 12;
```

```
      s := lst[Random(Length(lst))];
      th := Canvas.TextHeight(s);
      tw := Canvas.TextWidth(s);
      Canvas.TextOut(Xmarker - (tw div 2), ARect.Top + th, s);
    end;
  end;
```

Result:



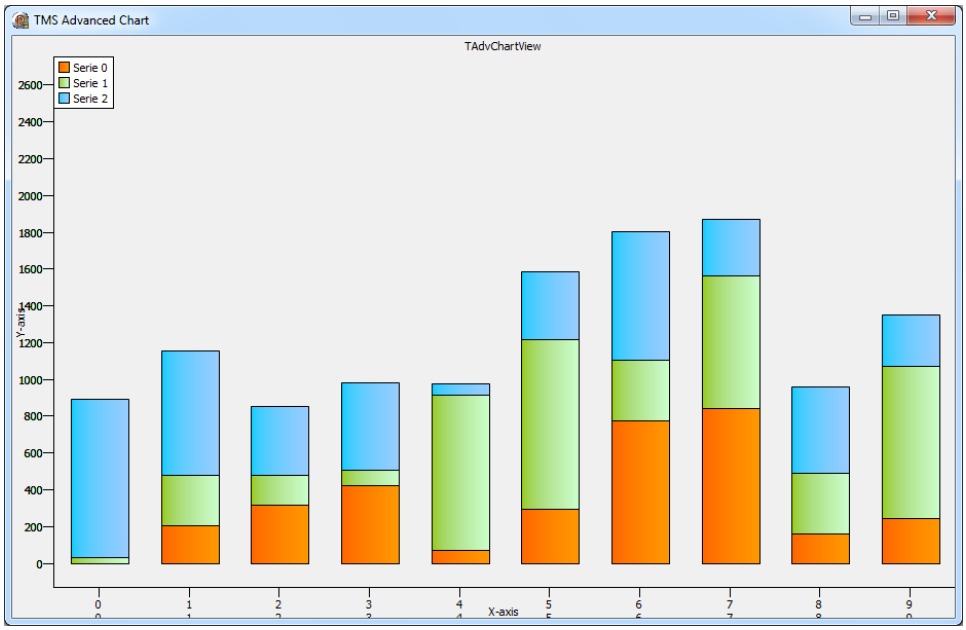
Different Bar Modes

Change different bar chart types to cylinder shape or pyramid shape:

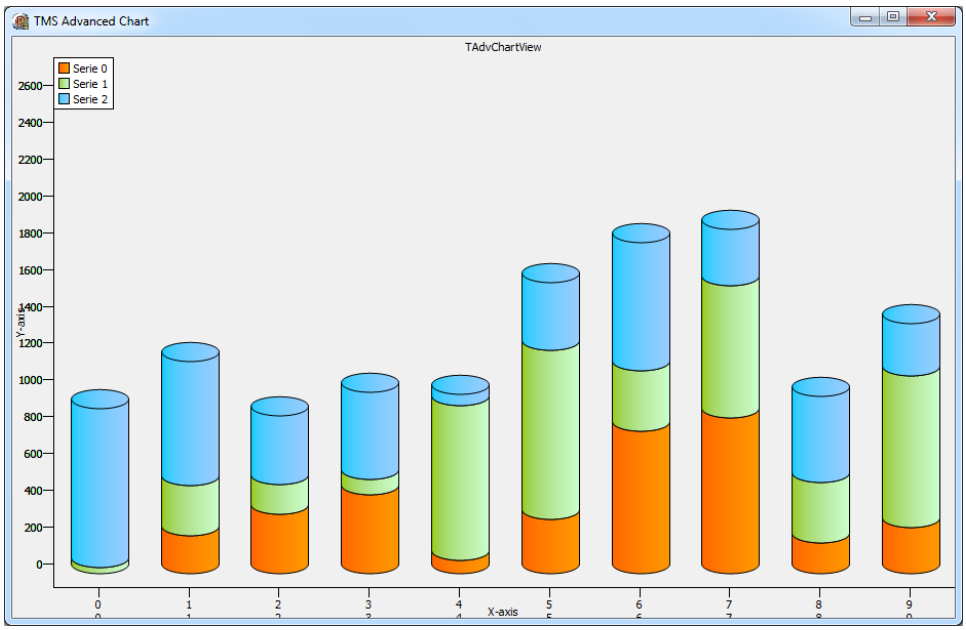
When adding new bar type charts the shape of the bar is bsRectangle by default. When changing the BarShape property on Series level the shape of the bar can be a Rectangle, Cylinder or a Pyramid.

Below is a sample that demonstrates the different shapes with one line of code for each series.

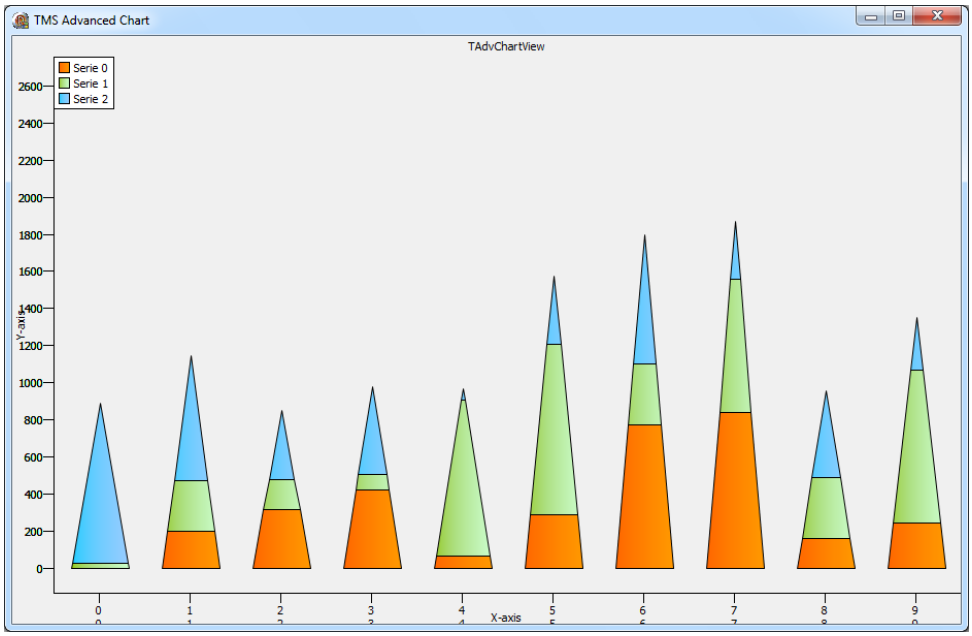
```
AdvGDIPChartView1.Panes[0].Series[0].BarShape := bsRectangle;
AdvGDIPChartView1.Panes[0].Series[1].BarShape := bsRectangle;
AdvGDIPChartView1.Panes[0].Series[2].BarShape := bsRectangle;
```




```
AdvGDIPChartView1.Panes[0].Series[0].BarShape := bsCylinder;
AdvGDIPChartView1.Panes[0].Series[1].BarShape := bsCylinder;
AdvGDIPChartView1.Panes[0].Series[2].BarShape := bsCylinder;
```




```
AdvGDIPChartView1.Panes[0].Series[0].BarShape := bsPyramid;
AdvGDIPChartView1.Panes[0].Series[1].BarShape := bsPyramid;
AdvGDIPChartView1.Panes[0].Series[2].BarShape := bsPyramid;
```



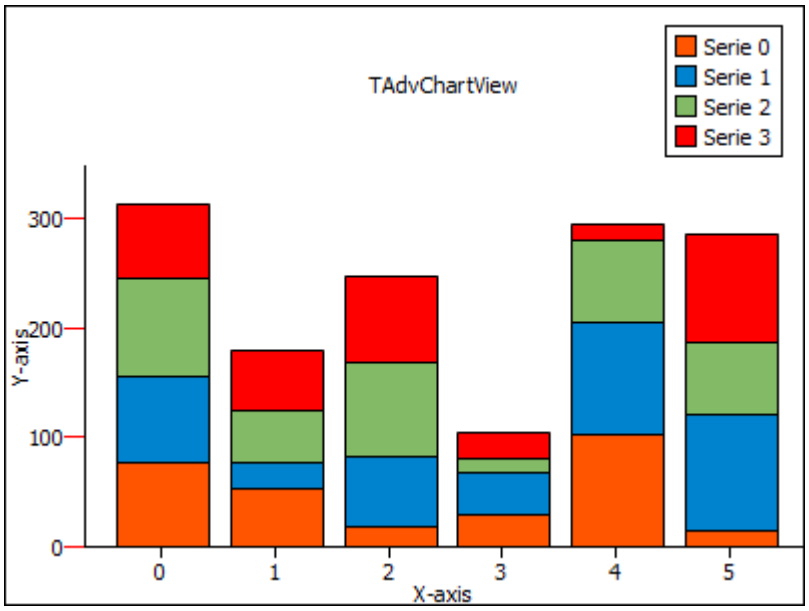
Grouped Stacked Bars

Each series has a GroupIndex property that can be used to group stacked bars. By default the groupindex property is 0 for all series. Each series is drawn on top of each other creating only one group. Below are some samples which demonstrate this property.

GroupIndex 0 for Series 1, 2, 3 and 4

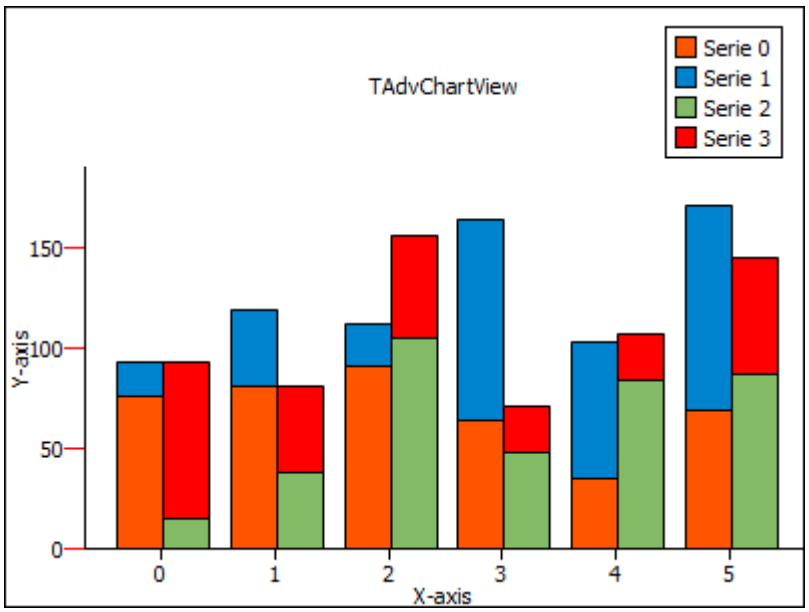
```
var
  I, K: integer;
  serie: TChartSerie;
begin
  AdvGDIPChartView1.BeginUpdate;
  AdvGDIPChartView1.InitSample;
  AdvGDIPChartView1.Panes[0].Series.Add;
  AdvGDIPChartView1.Panes[0].Legend.Left := 280;
  AdvGDIPChartView1.Panes[0].Legend.Top := -80;
  AdvGDIPChartView1.Panes[0].Title.Size := 80;
  for I := 0 to 3 do
  begin
    serie := AdvGDIPChartView1.Panes[0].Series[I];
    serie.ChartType := ctStackedBar;
    serie.AutoRange := arCommonZeroBased;
    serie.XAxis.Visible := I = 0;
    serie.YAxis.Visible := I = 0;
    serie.Marker.MarkerType := mNone;
    serie.ValueWidth := 100;
    serie.ClearPoints;

    for K := 0 to 5 do
    begin
      serie.AddSinglePoint(Random(100) + 10);
    end;

  end;
```

```
AdvGDIPChartView1.EndUpdate;
```



Changing the GroupIndex property of the last 2 series to 1 groups the first 2 series and the last 2 series.

```
if I > 1 then
  serie.GroupIndex := 1;
```



There are various possibilities to create grouped stacked bars. There is one rule to follow: The groupindex property must be increased while the series index increases. Adding a groupindex of 4 to the first series and an index of 2 to the second series will not work.

Pie, Half-Pie / Donut, Half-Donut

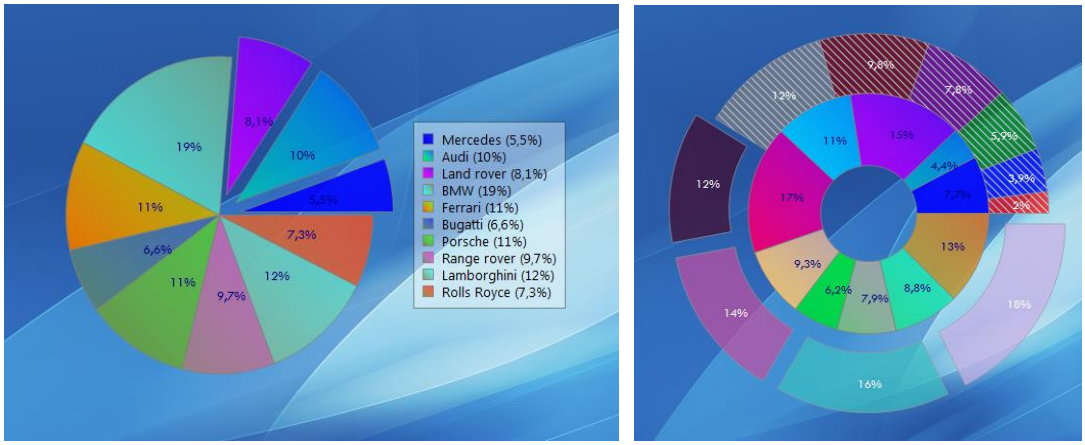
Add pie, half-pie / donut, half-donut with random values, exploded slices and legend:

Add a new AdvChartView and add points with the AddPiePoints method. Many extra parameters can be given to change the look of the pie chart such as exploded pie slices, undefined pie slices,… Many other options can be set in the Series Editor or via the chartserie properties. Note that the donut chart can also be displayed as stacked series. To enable this, set donut mode to stacked on Pane level.

Example:

```
const
  colors1: array[0..4] of TColor = (clRed, clBlue, clGreen, clPurple,
clMaroon);
  colors2: array[0..4] of TColor = (clDkGray, clLtGray, clLime,
clOlive, clLime);
  colors3: array[0..4] of TColor = (clBlue, clLime, clFuchsia,
clAqua, clRed);
  values: array[0..9] of String = ('Mercedes', 'Audi', 'Land rover',
'BMW', 'Ferrari', 'Bugatti', 'Porsche', 'Range rover', 'Lamborghini',
'Rolls Royce');
var
  I: Integer;
begin
  for I := 0 to 9 do
  begin
    Randomize;
    if I <= 2 then
AdvGDIPChartView1.Panes[0].Series[0].AddPiePoints(RandomRange(20,
100), values[I], colors1[1], Colors3[I], 20, true)
    else
AdvGDIPChartView1.Panes[0].Series[0].AddPiePoints(RandomRange(20,
100), values[I], clNone, clNone, 0, true);

  end;
```

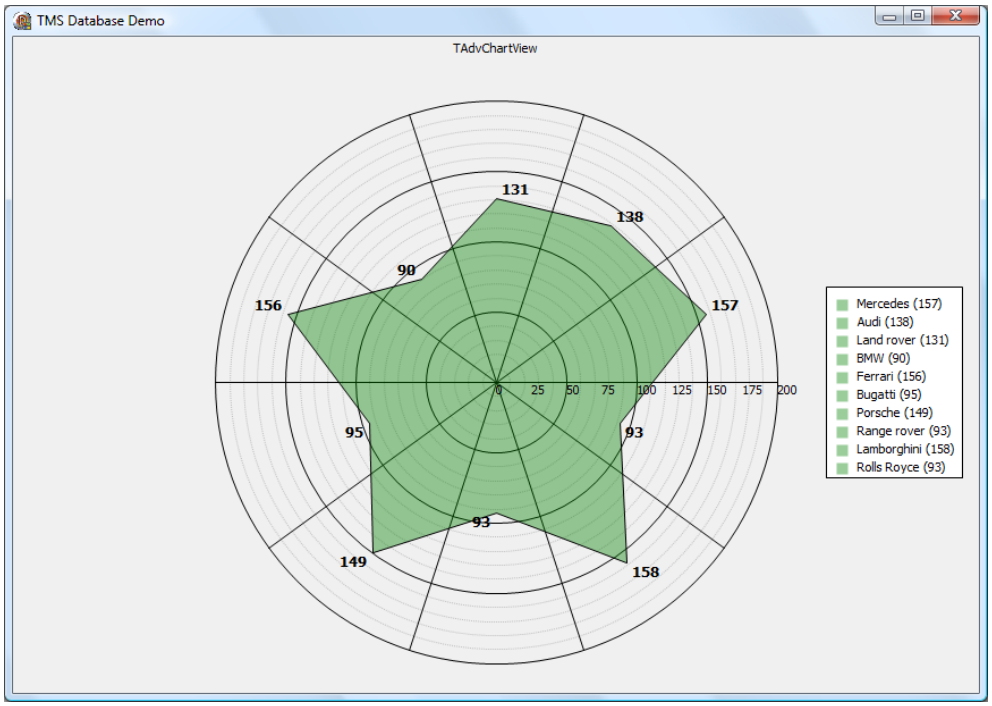
Result:



Spider / Halfspider

This is a sample that demonstrates the use of the spider chart. The spider chart is similar to an area chart that is stretched in a circular way. You can optionally enable the display of a grid and values.

```
procedure TForm1.FormCreate(Sender: TObject);
const
  values: array[0..9] of String = ('Mercedes', 'Audi', 'Land rover',
'BMW', 'Ferrari', 'Bugatti', 'Porsche', 'Range rover', 'Lamborghini',
'Rolls Royce');
var
  c: TColor;
  I: Integer;
begin
  for I := 0 to 9 do
  begin
    Randomize;
    with AdvGDIPChartView1.Panes[0] do
    begin
      YGrid.MajorLineColor := clBlack;
      YGrid.MinorLineColor := clSilver;
      YAxis.AutoUnits := false;
      XAxis.Position := xNone;
      YAxis.Position := yNone;
      Legend.Visible := false;
      with Series[0] do
      begin
        ChartType := ctSpider;
        YGrid.MajorDistance := 50;
        YGrid.MinorDistance := 10;
        YAxis.MajorUnit := 50;
        YAxis.MinorUnit := 25;
        LineColor := clBlack;
        Color := clGreen;
        Opacity := 100;
        Pie.Size := 400;
        Pie.ShowGridPieLines := true;
        Pie.ShowValues := true;
        Pie.ValueFont.Size := 10;
        Pie.ValueFont.Style := [fsBold];
        c := AdvGDIPChartView1.Panes[0].Series[0].Color;
        AddPiePoints(RandomRange(70, 200), values[I], c, c);
      end;
    end;
  end;
end;
```

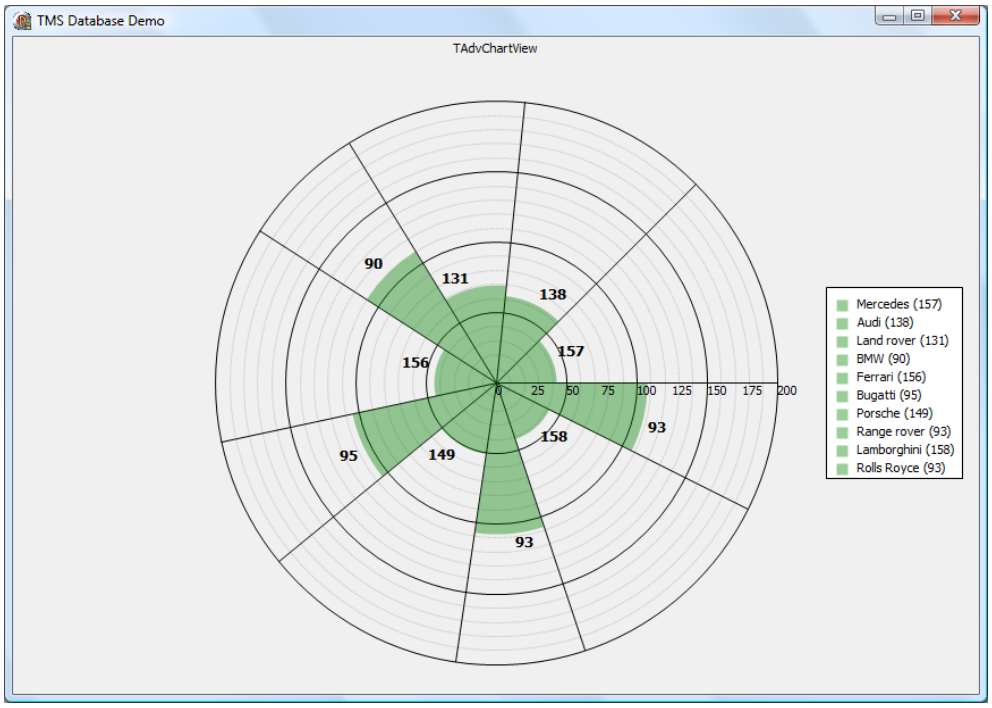


Variable Radius / Sized Pie

The variable radius pie has an extra parameter for the AddPiePoints method. The RadiusValue parameter allows you to specify the radius of each separate value.
In this sample the second radius parameter is inversely proportional with the added value. The larger the value, the smaller the radius will be.

```
procedure TForm1.FormCreate(Sender: TObject);
const
  values: array[0..9] of String = ('Mercedes', 'Audi', 'Land rover',
'BMW', 'Ferrari', 'Bugatti', 'Porsche', 'Range rover', 'Lamborghini',
'Rolls Royce');
var
  c: TColor;
  I, v: Integer;
begin
  for I := 0 to 9 do
  begin
    Randomize;
    with AdvGDIPChartView1.Panes[0] do
    begin
      YGrid.MajorLineColor := clBlack;
      YGrid.MinorLineColor := clSilver;
      YAxis.AutoUnits := false;
      XAxis.Position := xNone;
      YAxis.Position := yNone;
      Legend.Visible := false;
      with Series[0] do
      begin
        ChartType := ctVarRadiusPie;
        YGrid.MajorDistance := 50;
        YGrid.MinorDistance := 10;
        YAxis.MajorUnit := 50;
        YAxis.MinorUnit := 25;
```

```
        LineColor := clBlack;
        Color := clGreen;
        Opacity := 100;
        Pie.Size := 400;
        Pie.ValuePosition := vpOutSideSlice;
        Pie.ShowGridPieLines := true;
        Pie.ShowValues := true;
        Pie.ValueFont.Size := 10;
        Pie.ValueFont.Style := [fsBold];
        c := AdvGDIPChartView1.Panes[0].Series[0].Color;
        v := RandomRange(70, 200);
        AddPiePoints(v, 200 - v, values[I], c, c);
      end;
    end;
  end;
end;
```



With the Sized pie, the radius is calculated automatically, giving the largest value the largest radius. The angles of the slices are all the same.

Display of series values

Display values on chart points:

It is possible to display the values of chart series points at each series point by using property ShowValue on series level. Additional properties ValueFormat, ValueFont, ValueAngle, ValueType control how this value is displayed:

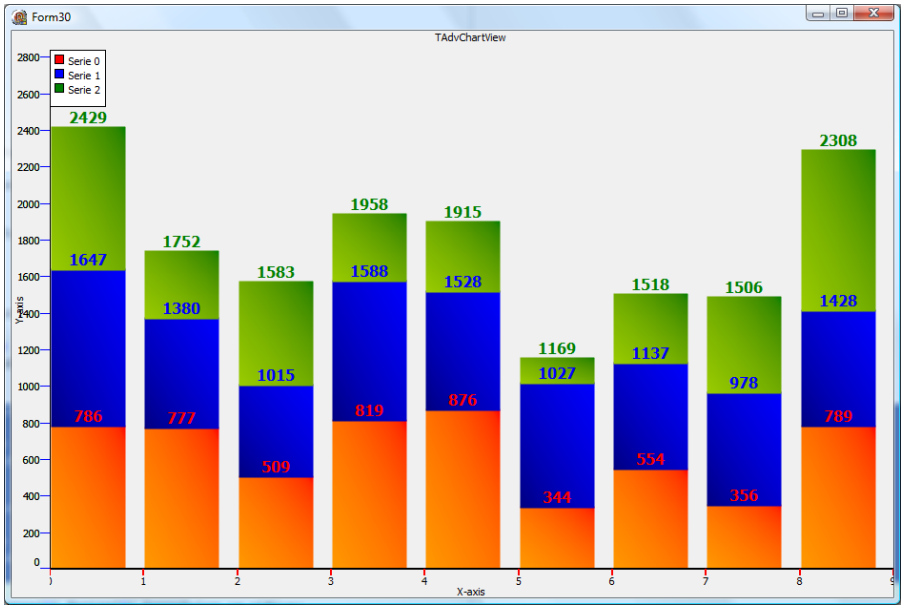
```
AdvChartView1.Panes[0].Series[0].ShowValue := true;
AdvChartView1.Panes[0].Series[0].ValueFont.Size := 12;
AdvChartView1.Panes[0].Series[0].ValueFormat := '%g';
AdvChartView1.Panes[0].Series[0].ValueAngle := 45;
AdvChartView1.Panes[0].Series[0].ValueType := cvNormal;
```

When using stacked bars / areas it can be useful if the total of each Bar / Area point is automatically calculated. With

```
AdvChartView1.Panes[0].Series.SerieValueTotals := true;
```

this is enabled.

The value of bar 2 is automatically added to the value of bar 1 (sample below)



Display values on Pie slices:

For pie / donut / spider chart types, the properties ValueType, ValueFormat also apply to enable the display of the series value on each pie slice:

```
AdvChartView1.Panes[0].Series[0].ValueType := cvNormal;
AdvChartView1.Panes[0].Series[0].ValueFormat := '%g';
```

Via the Pie property of the series you can configure the font size and color, the position of the values and determine whether the labels are visible or not:

```
AdvGDIPChartView1.Panes[0].Series[0].Pie.ValuePosition :=
vpOutSideSlice;
AdvGDIPChartView1.Panes[0].Series[0].Pie.ShowValues := true;
AdvGDIPChartView1.Panes[0].Series[0].Pie.ValueFont.size := 10;
```

By default the slice labels will be values only, and with:

```
AdvGDIPChartView1.Panes[0].Series[0].Pie.ShowLegendOnSlice := true;
```

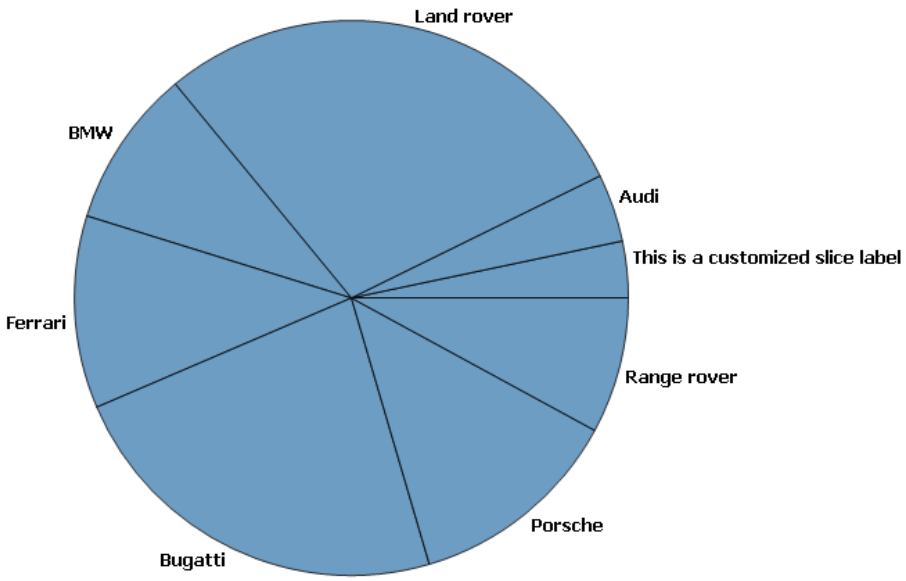
the legend text will be added on the slice label.

When needed, the pie slice labels can be further customized via the event OnSerieSliceGetValue and change the slice label text.

Sample:

```
AdvGDIPChartView1.Panes[0].Series[0].OnSerieSliceDrawValue :=
Serieslicedrawvalue;

procedure TForm35.Serieslicedrawvalue(Sender: TObject; serie:
integer; legendtext: String; value: Double; valuepointindex:
integer;
  var valueposition: TPoint; var valuestring: String);
begin
  if valuepointindex = 0 then
    valuestring := 'This is a customized slice label'
  else
    valuestring := legendtext;
end;
```

Result:

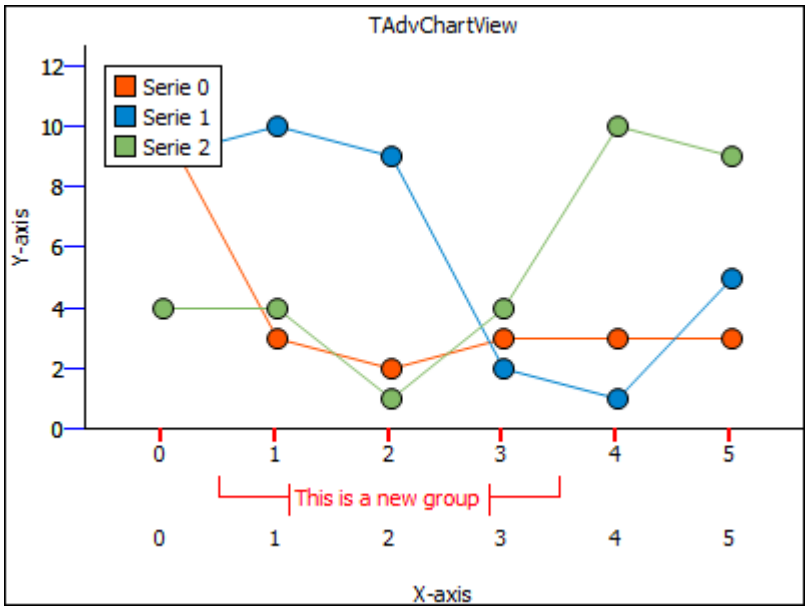


X-Axis grouping

Adding groups to the X-Axis

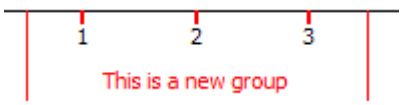
Each series has a XAxisGroups property that holds a collection of groups. A group is an indication / extra information of a set of points. To add a group add the following code:

```
var
  xGroup: TChartSerieXAxisGroup;
begin
  AdvGDIPChartView1.BeginUpdate;
  AdvGDIPChartView1.InitSample;
  AdvGDIPChartView1.Panes[0].XAxis.AutoSize := True;
  xGroup := AdvGDIPChartView1.Panes[0].Series[0].XAxisGroups.Add;
  xGroup.Caption := 'This is a new group';
  xGroup.StartIndex := 1;
  xGroup.EndIndex := 4;
  xGroup.LineColor := clRed;
  xGroup.Font.Color := clRed;
  xGroup.LineType := xgltWrap;

  AdvGDIPChartView1.EndUpdate;
```



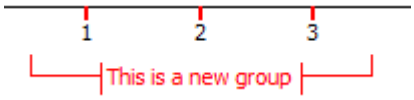
You can add multiple groups per series. Each group is automatically positioned under the X-Axis values. The line type can be changed to a different appearance.

xgltVertical:



xgltVerticalLine: The same as Vertical but if the group is on the second line of X-Axis values (second series) the vertical lines are drawn to the top of the X-Axis.

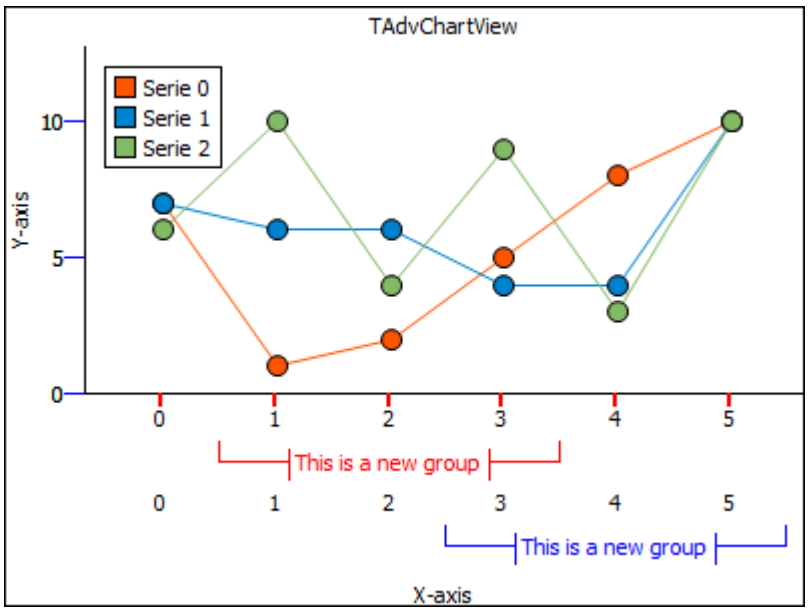
xgltWrap:



xgltCustom: Triggers a custom drawing event with a Rectangle, Serie and GroupIndex property.

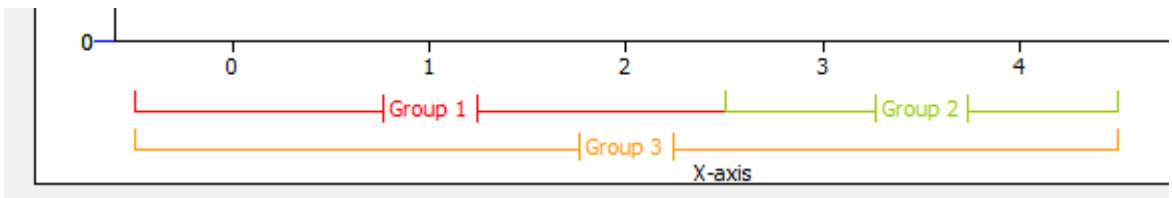
Adding a second group for the second series will result in a group that will be added under the second line of X-Values.

```
xGroup := AdvGDIPChartView1.Panes[0].Series[2].XAxisGroups.Add;
xGroup.Caption := 'This is a new group';
xGroup.StartIndex := 3;
xGroup.EndIndex := 6;
xGroup.Font.Color := clBlue;
xGroup.LineColor := clBlue;
xGroup.LineType := xgltWrap;
```



Multiple groups can be added per series, and also have a different level. Adding 3 groups with respectively level 0, 0 and 1 will result in the following structure:

```
xGroup1.Level := 0;
xGroup2.Level := 0;
xGroup3.Level := 1;
```



TChartAnnotation

General overview

Adding and removing annotations: Start the AnnotationEditorDialog and click on the "-" sign to remove or the "+" sign to add an annotation. You can add multiple annotations per series. Here is an example to remove an annotation and to add an annotation to the first series of the first pane.

```
procedure TForm1.AddNewAnnotation;
var
  ca: TChartAnnotation;
begin
  ca := AdvChartView.Panes[0].Series[0].Annotations.Add;
  // set properties of the chart annotation here
end;

procedure TForm1.RemoveAnnotation;
begin
  AdvChartView.Panes[0].Series[0].Annotations[0].Free;
end;
```

Note: for pane, series and annotation adding create use the following code to set properties.
Example with annotations:

```
procedure TForm1.AddNewAnnotation;
begin
 with AdvChartView.Panes[0].Series[0].Annotations.Add do
 begin
  // ADD ALL CODE TO SET PROPERTIES HERE EXAMPLE:
  Shape := asBalloon;
 end;
end;
```

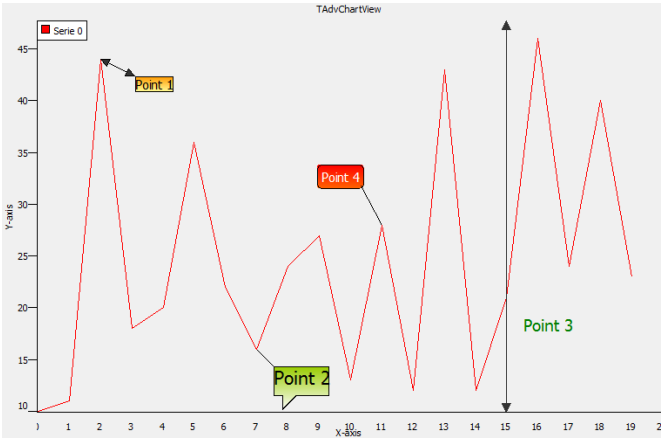
Different kinds of annotations

There are different kinds of annotations that can be used to mark special points of interest in a series.

Balloon, Circle, RoundRectangle, Rectangle : Shapes to wrap around the text.
Line: a line drawn from top to bottom on the X-position of the point.

Offset (X, Y): the text offset from the X, Y- position of the point.

Here is a screenshot with the different shapes.



TAdvGDIPChartView

TMS VCL Chart also comes with a GDI+ version of TAdvChartView and adds extra features such as anti-aliased drawing, transparency, a special zoom control window, and much more.

Zoom Control Window

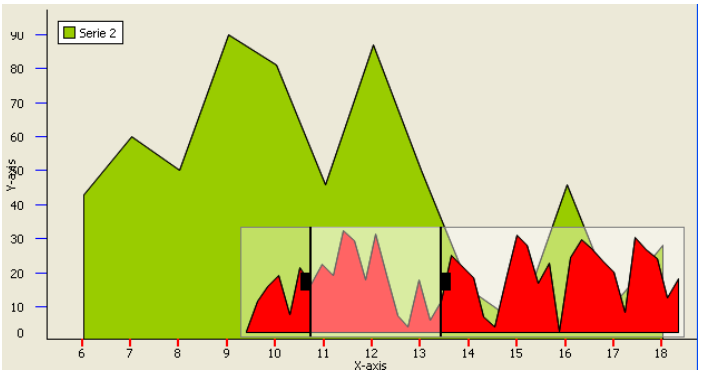
The zoom control window allows the user to 'navigate' through the chart and always displays the complete chart range by default. There are 2 handles that can be dragged left or right representing the RangeFrom and RangeTo of the Chart. The zoom control appearance can be customized in the GDI+ Panes Editor Dialog.

When you start the application, the zoomcontrol window will be empty. Therefore it is important to link the series by setting the SerieType property of TAdcChartSerie. The SerieType property specifies how and what series from the series collection of the chart is used on the zoomcontrol.

stNormal: This is the default value, the series will not be visible in the zoom control window.
stZoomControl: The series will only be displayed in the zoom control window.
stBoth: The series will be displayed in the zoom control window and as a normal series.

By default the chart will be automatically updated when changing the range. The property AutoUpdate can be changed to update immediate, upon release of the mouse button, or no automatic update. When no automatic update is set, you can use the 3 events below:

OnZoomControlRangeChanged: Event triggered when Range is updated after mouse button release
OnZoomControlRangeChanging: Event triggered when Range is updated when mouse is moving
OnZoomControlSelection: Event triggered when Range is updated after dragging the Selection area between the range from and range to.



TDBAdvChartView & TDBAdvGDIPChartView Basics

TMS VCL Chart has the ability to link with databases and automatically add points from record fields. To do this, the DB-aware chart can be linked to a dataset via a datasource and the dataset field can be choosen for the values of a series and optionally a different dataset field can be choosen as X-axis values. The DB-aware chart will automatically load values of all rows in the dataset as series data points and it will also automatically update values when the dataset is being edited or when new rows are inserted in the dataset.

Below is a complete sample which demonstrates the chart-database link, the different fields...

Connecting DataSource to the Chart

Drop a TDBAdvGDIPChartView on the form. When double-clicking the chart view, the panes editor dialog shows and a new tabsheet named “Database”.

Drop a TTable component on the form. Choose DBDemos as database and employees.db as tablename. Then link the TTable to the TDataSource by choosing Table1 as Dataset.

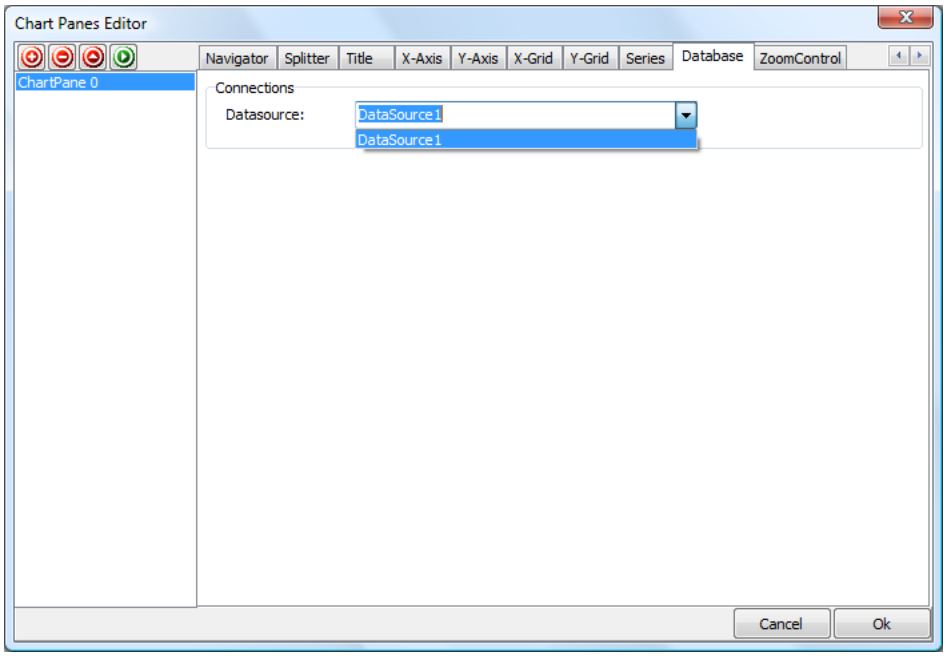
Table1

Filtered	customer.db
FilterOptions	employee.db
IndexDefs	events.db
IndexFieldNames	holdings.dbf
IndexFiles	industry.dbf
IndexName	items.db
MasterFields	master.dbf
MasterSource	nextcust.db
Name	nextitem.db
ObjectView	nextord.db
Ranged	orders.db
ReadOnly	parts.db
SessionName	reservat.db
StoreDefs	vendors.db
TableName	venues.db
	employee.db

DataSource1

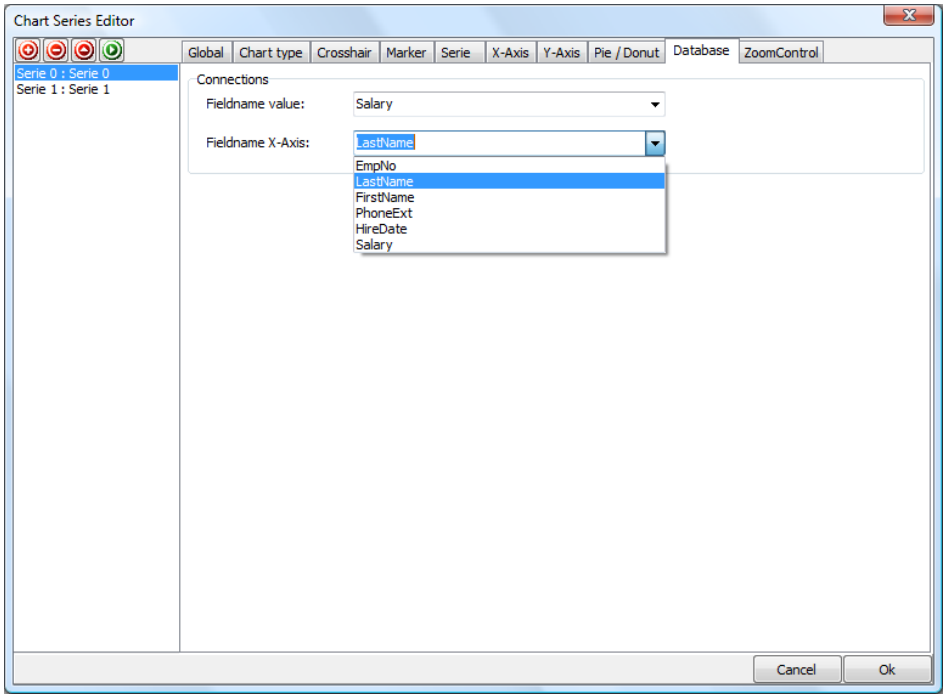
AutoEdit	True
DataSet	**Table1**
Enabled	Table1
Name	DataSource1
Tag	0

Double-click on the chart to start the panes editor dialog. Go to the TabSheet “Database” and select DataSource1 as DataSource property. Return to the form and set Active to true on Table1 dataset.



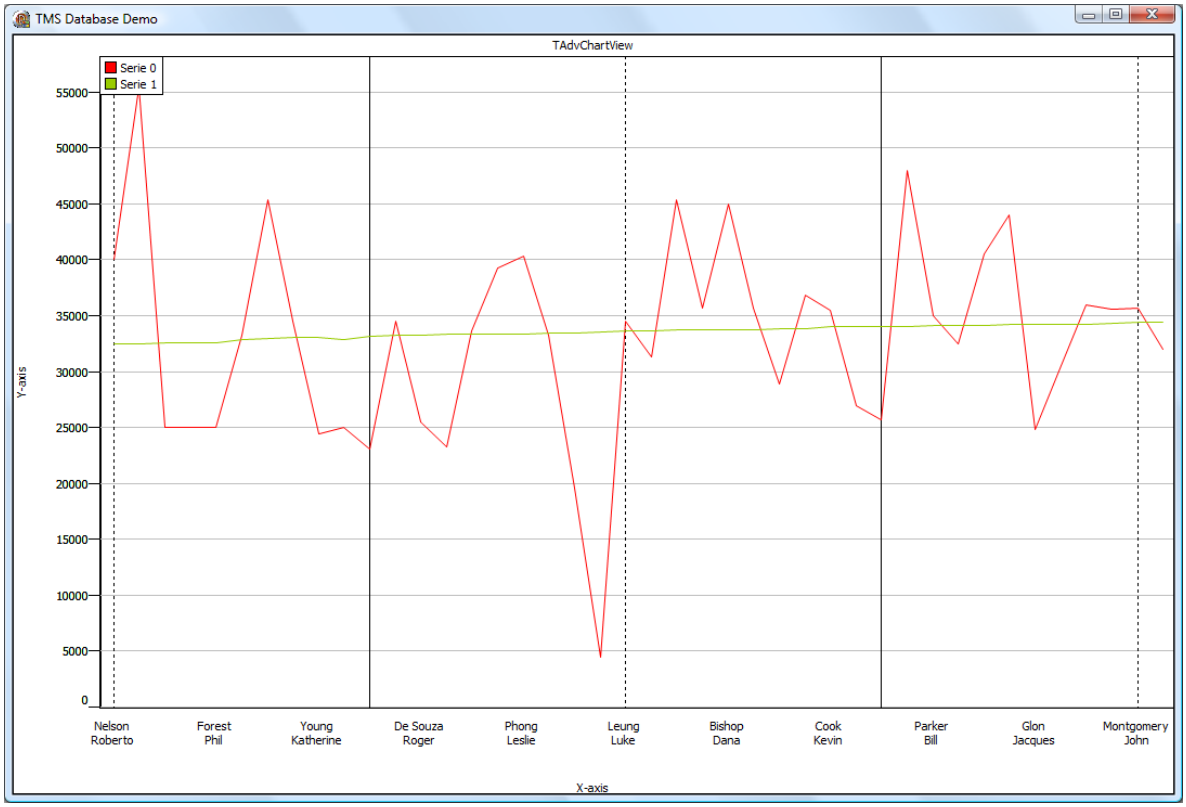
Connecting Fields to Series

After setting the DataSource on the chart the series are aware of the selected tablename and will display the fields of the table employees.db



Remove the 2 other series and leave series 0.
Select Salary as Fieldname value for Series 0 and LastName as Fieldname X-Axis for Series 0.
After saving the changes, the chart will automatically be updated with every record that exists in the database. Choose a line type and change the AutoRange to arEnabled for automatic Y-Range.

Result


TMS Database Demo
TAdvChartView
Serie 0
Serie 1
55000
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
Y-axis
Nelson Roberto
Forest Phil
Young Katherine
De Souza Roger
Phong Leslie
Leung Luke
Bishop Dana
Cook Kevin
Parker Bill
Glon Jacques
Montgomery John
X-axis

Programmatic use of DBAdvChartView & TDBAdvGDIPChartView

Below is a code sample to show values from a Microsoft Access Database, using an AdoDataSet, AdoConnection and a Datasource.

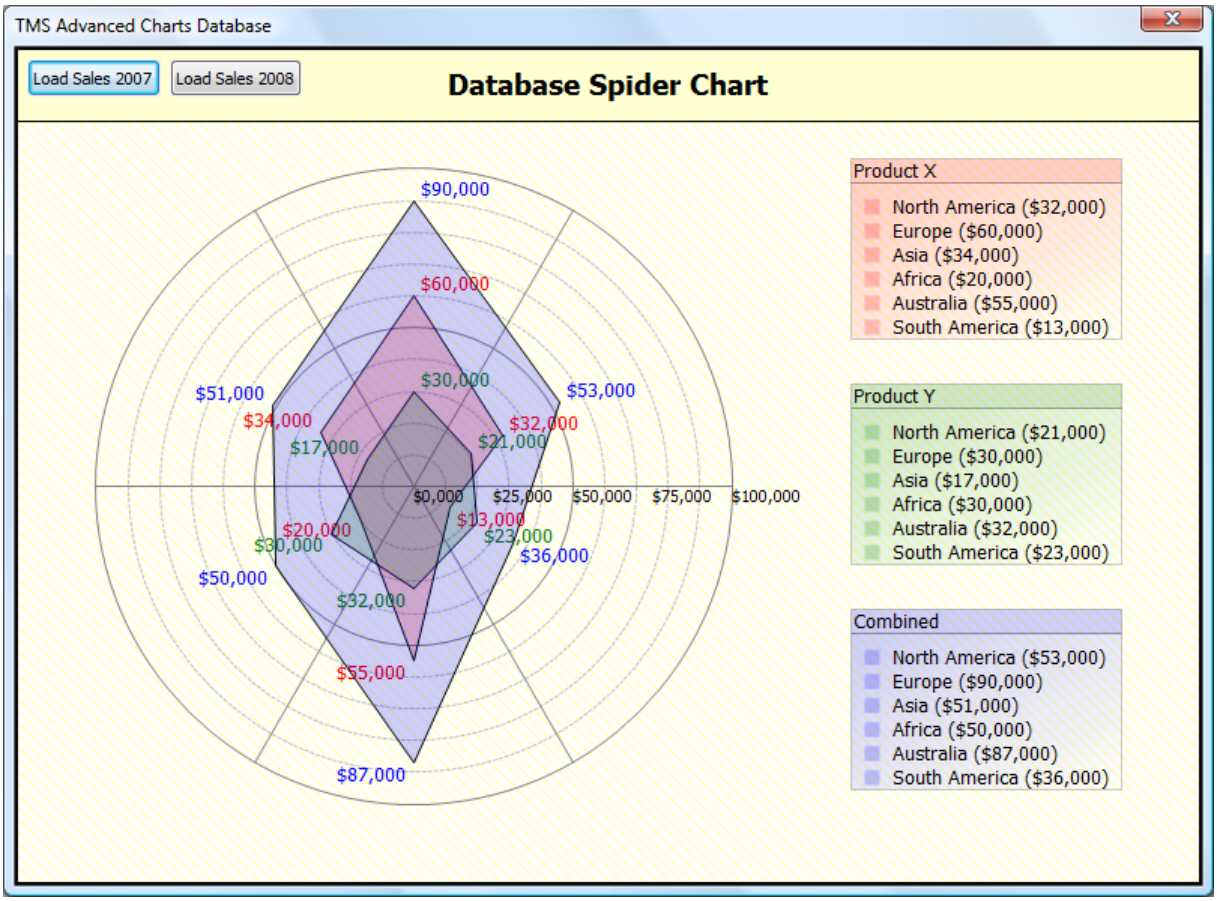
Drop a TDBAdvGDIPChartView, TAdoConnection, TAdoTable and a TDataSource on the form. Insert this code in the procedure FormCreate:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  I: integer;
begin
  DBAdvGDIPChartView1.Panes[0].BorderColor := clBlack;
  DBAdvGDIPChartView1.Panes[0].BorderWidth := 3;

  with DBAdvGDIPChartView1.Panes[0] do
  begin
    Series.Clear;
    //Set connection string to database
    ADOConnection1.ConnectionString :=
'Provider=Microsoft.Jet.OLEDB.4.0;Data Source=sales.mdb;Persist Security
Info=False';
    ADOConnection1.Connected := true;

    //Set tablename
    ADOTable1.Connection := ADOConnection1;
    ADOTable1.TableName := 'Sales';

    //link to Datasource and datasource to chart
    DataSource1.DataSet := ADOTable1;
    //Datasource property of Chart
    DataSource := DataSource1;

    XAxis.Position := xNone;
    YAxis.Position := yNone;
    Title.Color := RGB(255, 255, 210);
    Title.ColorTo := RGB(255, 255, 210);
    Title.BorderColor := clBlack;
    Title.BorderWidth := 1;
    Title.GradientDirection := cgdVertical;
    Title.Size := 50;
    Title.Text := 'Database Spider Chart';
    title.Font.Size := 14;
    Title.Font.Style := [fsBold];
    Background.GradientType := gtHatch;
    Background.HatchStyle := HatchStyleWideDownwardDiagonal;
    Background.Color := RGB(255, 255, 210);
    BackGround.ColorTo := clWhite;
    Legend.Visible := false;
    Margin.LeftMargin := 0;
    Margin.rightMargin := 0;
    Margin.TopMargin := 0;
    series.DonutMode := dmStacked;

    YAxis.AutoUnits := false;
    YGrid.MajorDistance := 50;
    YGrid.MinorDistance := 10;
    YGrid.MinorLineColor := clSilver;
```

```
YGrid.MajorLineColor := clDkGray;
YGrid.MinorLineStyle := psDash;

Series.Add;

with Series[0] do
begin
  Pie.LegendOffsetTop := (Self.Height div 2) - 150;
  Pie.LegendColor := clRed;
  Color := clRed;
  FieldNameValue := 'Product X';
  FieldNameXAxis := 'Sales by Region';
  LegendText := 'Product X';
  Pie.LegendTitleColor := clRed;
  Pie.ValueFont.Color := clRed;
end;

Series.Add;

with Series[1] do
begin
  Pie.LegendOffsetTop := (Self.Height div 2);
  Pie.LegendColor := clGreen;
  Color := clGreen;
  Pie.ShowGrid := false;
  YAxis.Visible := false;
  FieldNameValue := 'Product Y';
  FieldNameXAxis := 'Sales by Region';
  LegendText := 'Product Y';
  Pie.LegendTitleColor := clGreen;
  Pie.ValueFont.Color := clGreen;
end;

Series.Add;

with Series[2] do
begin
  Pie.LegendOffsetTop := (Self.Height div 2) + 150;
  Pie.LegendColor := clBlue;
  Color := clBlue;
  Pie.ShowGrid := false;
  YAxis.Visible := false;
  FieldNameValue := 'Combined';
  FieldNameXAxis := 'Sales by Region';
  LegendText := 'Combined';
  Pie.LegendTitleColor := clBlue;
  Pie.ValueFont.Color := clBlue;
end;

for I := 0 to Series.Count - 1 do
begin
  with Series[I] do
  begin
    YAxis.MajorUnit := 50;
    YAxis.MajorUnit := 25;
    Pie.LegendTitleVisible := true;
    pie.LegendOpacity := 50;
    pie.LegendOpacityto := 0;
```

```
        Pie.LegendTitleOpacity := 50;
        Pie.LegendGradientType := gtForwardDiagonal;
        Pie.LegendBorderColor := clBlack;
        Pie.Position := spCustom;
        Pie.Left := self.Width div 3;
        Pie.Top := self.Height div 2;
        Pie.LegendFont.Size := 10;
        ValueFormat := '$%g,000';
        Pie.ValueFont.Size := 10;
        ChartType := ctSpider;
        Opacity := 50;
        LineColor := clBlack;
        AutoRange := arCommonZeroBased;
        Pie.Size := 400;
        Pie.ShowValues := true;
        Pie.ValuePosition := vpOutSideSlice;
        pie.LegendPosition := spCustom;
        Pie.LegendOffsetLeft := Self.Width - 150;
      end;
    end;

    // open connection
    ADOTable1.Active := true;
  end;
end;
```

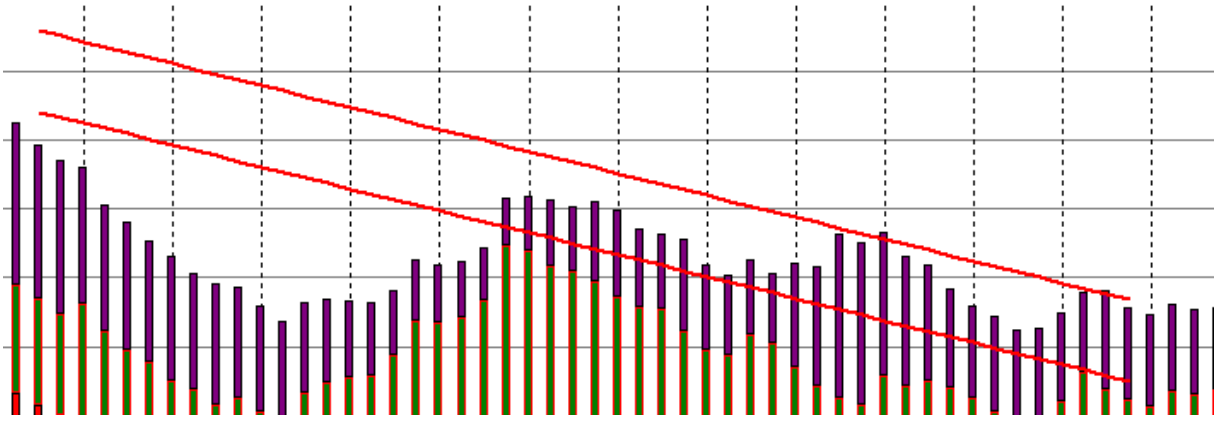
Result:



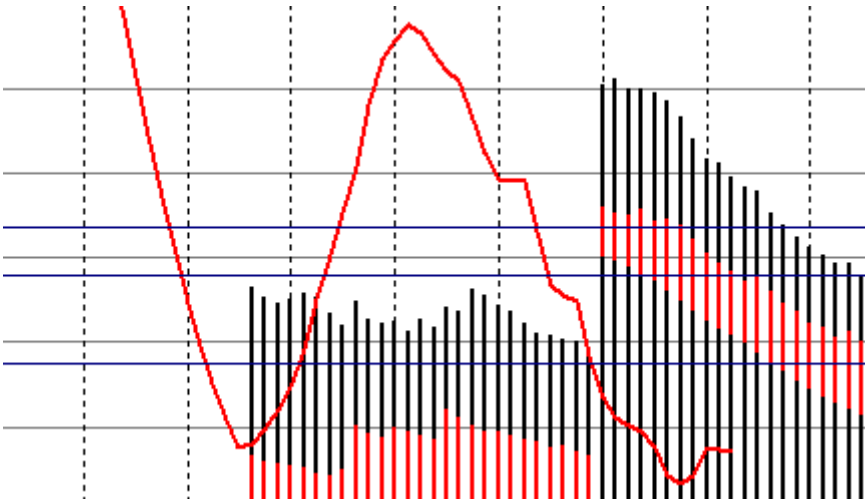
Adding Special Series

TMS VCL Chart allows you to add trend lines, moving averages, line series and band series.
This series is not visible in the legend and will be used as an extra "calculated data - series". These series are automatically calculated from specific values in a defined range from series available in the chart. As such, it is mandatory to first add the series values and when values are added, call the method to calculate the trend line series, moving averages series etc…

Trend Channel



Moving Average



Below are the method definitions that can be used to add informational series to the chart.

```
procedure AddLineSerie(startx, endx: integer; startvalue, endvalue:
  Double; UseCommonYRange: Boolean = true; serielinecolor: TColor =
  clBlack; serielinewidth: integer = 1);

procedure AddBandSerie(startx, endx: integer; startvalue1, startvalue2,
  endvalue1,endvalue2: Double; UseCommonYRange: Boolean = true;
  serielinecolor: TColor = clBlack; serielinewidth: integer =
  1;seriebandcolor: TColor = clGray);
```

```
procedure AddTrendLine(serieindex, startx, endx: integer;
  UseCommonYRange: Boolean = true; serielinecolor: TColor = clBlack;
  serielinewidth: integer = 1);

procedure AddTrendChannel(serieindex, startx, endx: integer;
  UseCommonYRange: Boolean = true; serielinecolor: TColor = clBlack;
  serielinewidth: integer = 1);

procedure AddTrendChannelBand(serieindex, startx, endx: integer;
  UseCommonYRange: Boolean = true; serielinecolor: TColor = clBlack;
  serielinewidth: integer = 1; seriebandcolor: TColor = clGray);

procedure AddMovingAverage(serieindex, startx, endx, window: integer;
  UseCommonYRange: Boolean = true; serielinecolor: TColor = clBlack;
  serielinewidth: integer = 1);

procedure AddParetoLine(serieindex, startx, endx: integer;
  UseCommonYRange: Boolean = true; serielinecolor: TColor = clBlack;
  serielinewidth: integer = 1);
```

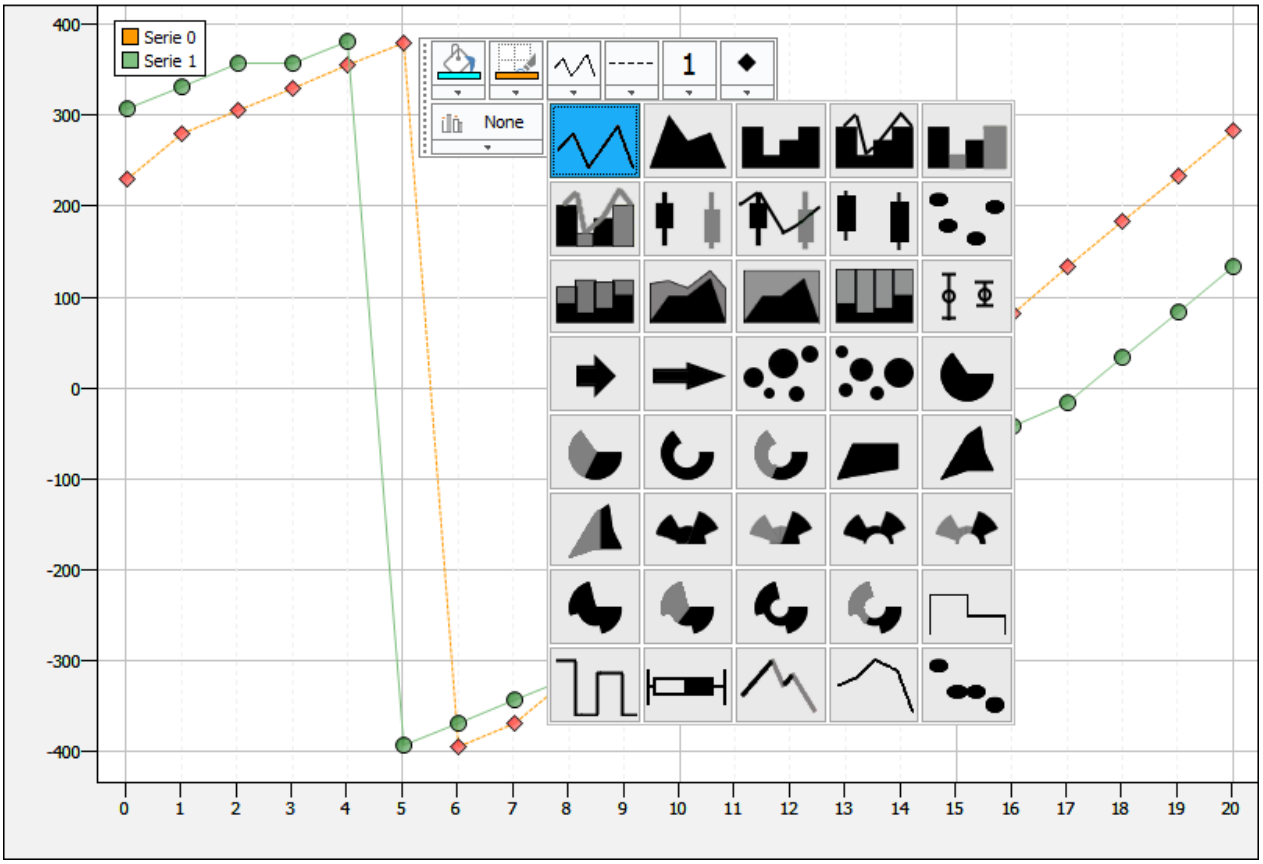
Popup ToolBar (XE2 and newer)

The chart supports displaying a helper popup toolbar that offers various functionality such as changing the series type, fill and line color as well a specifying the marker type and label visibility.

To enable the popup, set Popup.Enabled := True;

AdvChartView1.Popup.Enabled := True;

When clicking on a series point, slice, or bar depending on the chosen series type the popup will be shown and initialized based on the current settings of the series.

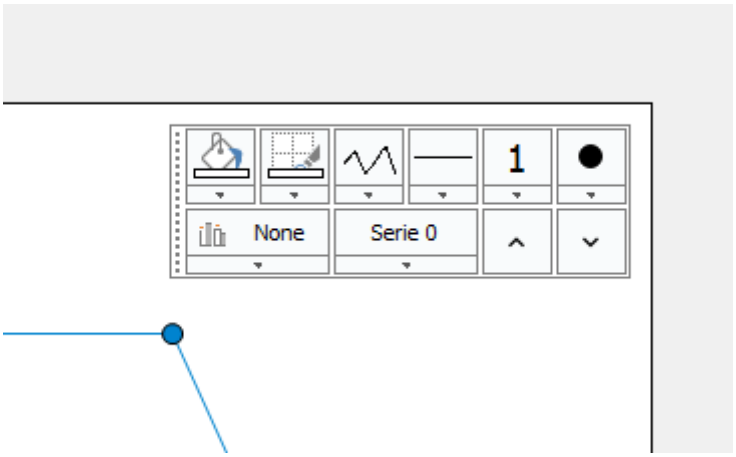


Changing the values in the dropdown pickers will change the appearance of the selected series. Clicking next to the point or on another area of the chart will close the popup. The popup options can be configured via the Popup.Options property. This allows you to show more or less dropdown pickers depending on the series configuration. Below is a sample that demonstrates displaying the full set of options versus a limited set with line color, type and width.

AdvChartView1.Popup.Options := AllOptions;

AdvChartView1.Popup.Options := [ctpoStroke, ctpoLineStyle, ctpoLineWidth];

The popup can be dragged to another position after showing it, to allow it to be positioned without overlapping the chart drawing. As soon as a new marker/point/slice is clicked, the popup will be repositioned. Hover the mouse over the draggable area at the left side of the popup, click and hold the left mouse button to reposition the popup.



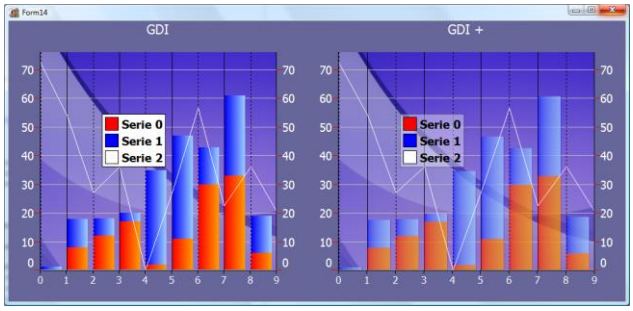
Programmatic access

The popup is accessible via the ToolBarPopup property. Further customizations related to appearance and functionality can be configured via this property. Adding new or existing controls to the popup can be done with one of the AddToolBarPopup* functions that are exposed at TAdvChartView or TAdvGDIPChartView level.

GDI vs GDI+

The size of GDIPLUS DLL is +/- 2MB. It is available by default in Windows Vista and Windows XP but not in Windows 98/Me/2000. This means that the GDI+ DLL should be included in application software distributions for OS versions older than Windows XP. Although the deployment of the GDIPLUS.DLL is as simple as copying it along the application executable some users will prefer to not use GDI+ and other will prefer to enjoy the various benefits of GDI+ like anti aliasing, transparencies, complex gradients, hatches,… Therefore we have created a standard GDI based TAdvChartView and a more graphically advanced TAdvChartViewGDIP component.

The GDI+ version of the AdvChartView currently supports gradient colors with complex color gradients with optional configurable gradient angle, GDI+ hatch styles, opacity gradients, PNG image support with alpha transparency. These features apply for the annotations, markers, background, legend and bar, area, line, bubble chart types. Below you see a sample of 2 identical charts. One with GDI and the other with GDI+ features.



Single versus multi value charts and data per chart point

When you add single or multi points you can set some other properties such as color, timestamp, defined, open, high, low, close values. There are some chart types which cannot be used with the color property, chart types require only one value per point.

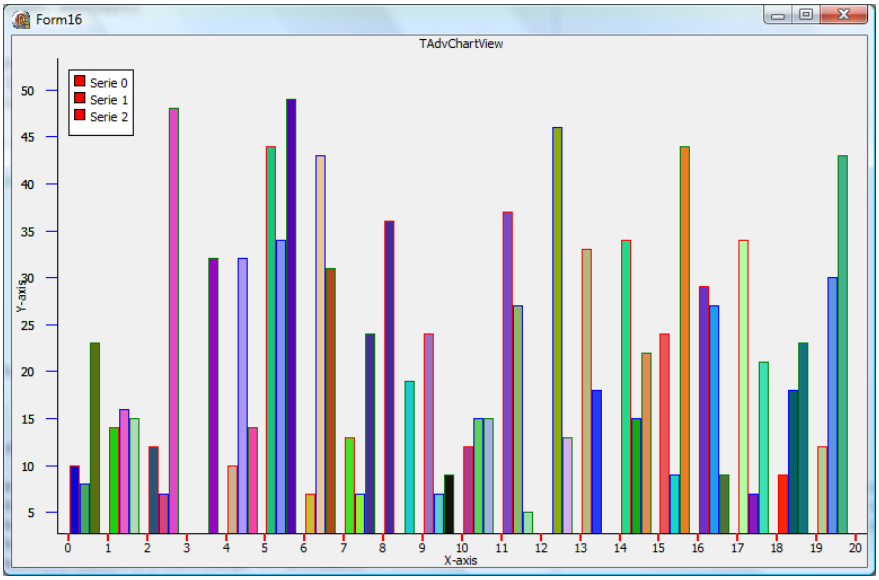
Here is an overview of chart types and what values / properties are supported:

Example: Histogram with colored bars.

The first sample is a histogram chart type which can be drawn with a single random color per point. Add to following code to the form create:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 20 do
  begin
    AdvChartView1.Panes[0].Series[0].AddSinglePoint(Random(50),
TColor(RGB(Random(255), Random(255), Random(255))));
    AdvChartView1.Panes[0].Series[1].AddSinglePoint(Random(50),
TColor(RGB(Random(255), Random(255), Random(255))));
    AdvChartView1.Panes[0].Series[2].AddSinglePoint(Random(50),
TColor(RGB(Random(255), Random(255), Random(255))));
    AdvChartView1.Panes[0].Series[0].AutoRange := arCommon;
    AdvChartView1.Panes[0].Series[1].AutoRange := arCommon;
    AdvChartView1.Panes[0].Series[2].AutoRange := arCommon;
  end;
end;
```

Result:

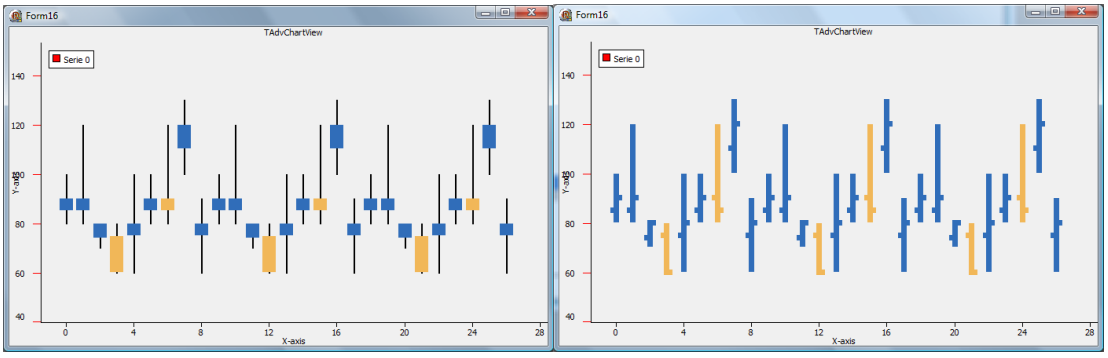


Example: Adding multiple values to use Candlestick / OHLC

Add open, high, low, close values to the Candlestick / OHLC chart type with the AddMultipoints procedure. Two extra properties can be set to select the increase and decrease color.

```
procedure TForm16.FormCreate(Sender: TObject);
begin
  Randomize;
  with AdvChartView1.Panes[0].Series[0] do
  begin
    AddMultiPoints(80, 70, 74, 80);
    AddMultiPoints(80, 60, 75, 60);
    AddMultiPoints(100, 60, 75, 80);
    AddMultiPoints(100, 80, 85, 90);
    AddMultiPoints(120, 80, 90, 85);
    {… and more}
    AutoRange := arDisabled;
    Maximum := 150;
    Minimum := 40;
    CandleColorDecrease := TColor(RGB(Random(255), Random(255),
Random(255)));
    CandleColorIncrease := TColor(RGB(Random(255), Random(255),
Random(255)));
  end;
end;
```

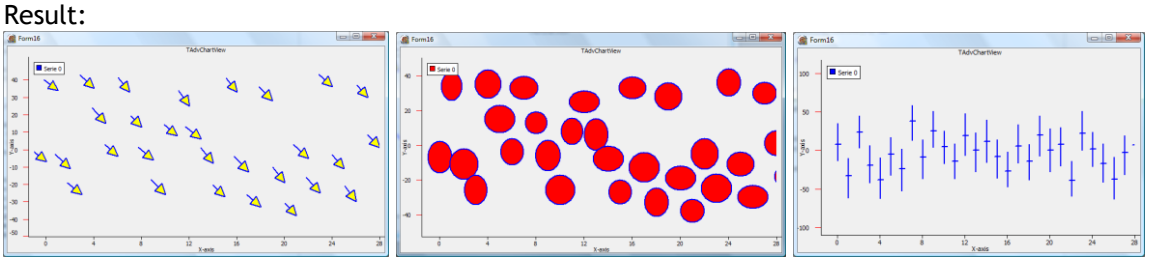
Result:



Example: Using optional value 1 and value 2 to draw error, arrow or bubble charts.

Use AddPoints where the first value is the position/value along the Y-axis, the second and the third are the optional values. When an error chart is chosen the optional value 1 is the value above and the optional value 2 below the Y-value. When an arrow chart is chosen, the optional value 1 / value 2 set the x, y delta from the actual chart X, Y point of the arrow start point. In the bubble chart the optional value 1 is the height and the optional value 2 is the width of the bubble.

Result:



Save and restore TAdvChartView settings

All properties of the TAdvChartView can be saved to a .Chart file. If you close the application and load the .Chart file, all settings will be restored. Here is an example of saving / reloading settings in code.

Save settings:

```
procedure TForm1.SaveToFile1Click(Sender: TObject);
begin
  if SaveDialog1.Execute then
  begin
    AdvChartView1.SaveToFile(SaveDialog1.FileName);
    ShowMessage('Settings Saved');
  end;
end;
```

Load settings:

```
procedure TForm1.LoadSettings1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
  begin
    AdvChartView1.LoadFromFile(OpenDialog1.FileName);
    ShowMessage('Settings Loaded');
  end;
end;
```

Printing

The support for printing in TAdvChartView is on level of panes. The chart can print a single pane or can print all panes at once. As the print function just sends the chart to a destination Canvas, the print function can be used to send the chart to a printer device but it can be used to send the output also to a memory bitmap canvas:

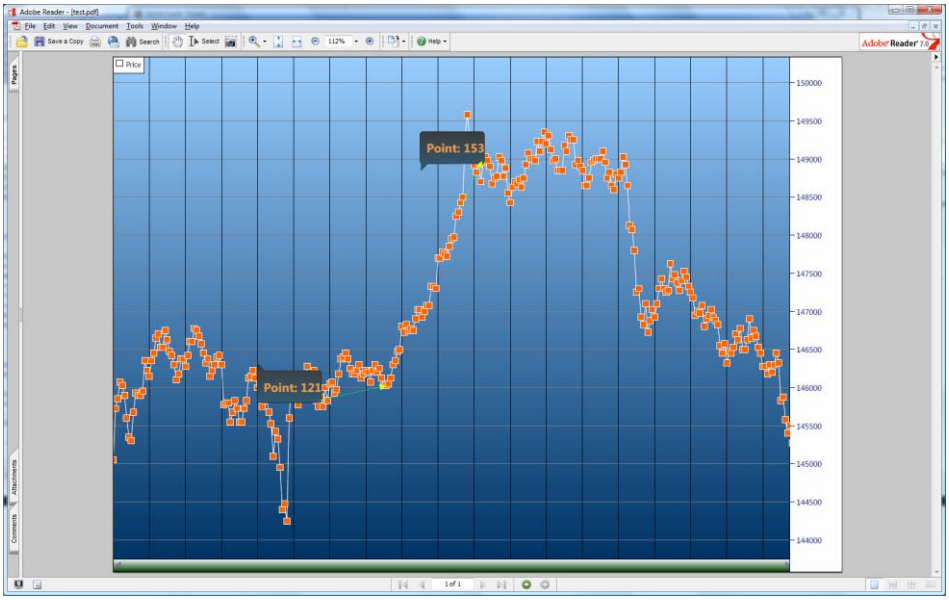
Print on a printer

```
if PrinterSetupDialog1.Execute then
begin
  with printer do
  begin
    BeginDoc;

    //The rectangle to print the AdvChartView.
    r := Rect(0, 0, printer.PageWidth, printer.PageHeight);

    AdvChartView.PrintAllPanes(Printer.Canvas, r);
    //OR print a single pane.
    AdvChartView.PrintPane(0, Printer.Canvas, r);

    EndDoc;
  end;
end;
```

Use the print function to save to a bitmap

```
if SavePictureDialog1.Execute then
begin
  //for example the size of the picture has been set to a 1280 x 1024
  //resolution.
  AdvChartView1.SaveAllPanesToBitmap(SavePictureDialog1.FileName ,1280,
  1024);
end;
```

Result in PDF file (Landscape):



Exporting to PNG, JPEG, TIFF, BMP or GIF files (GDI+)

Saving to a Windows bitmap file offers good quality but generates large files. You can save the AdvChartView to different image formats to improve / reduce the size and quality. The image formats currently supported are JPEG, PNG, BMP, GIF, TIFF. You can also save a single pane to a different image format. Using the method SaveToImage(), by default the image will be saved to Windows bitmap (BMP) format. If you set the extra format parameter (itJPEG, itBMP, itPNG, itGIF, itTIFF), you can choose in which encoded image type your result image will be saved.

Example:

Save to a jpeg with 50 % quality (quality only has effect on the jpeg format).

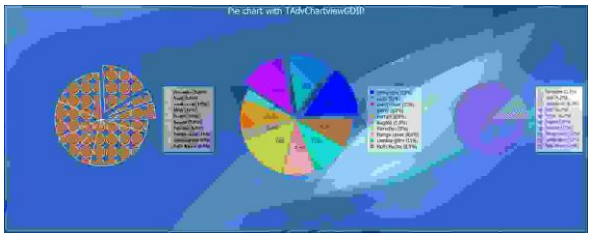
```
if SavePictureDialog1.Execute then
begin
  //for example the size of the picture has been set to a 1000 x 500
  //resolution.
  AdvChartView1.SaveToImage(SavePictureDialog1.FileName, 1000, 500, itJPEG,
  50);
end;
```

Save a single chart pane to a PNG format

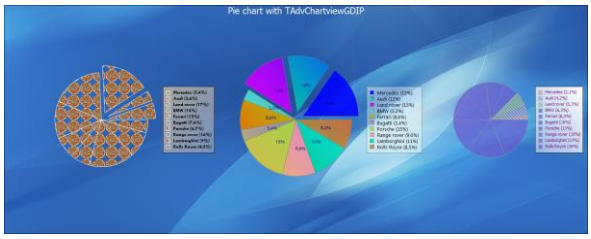
```
if SavePictureDialog1.Execute then
begin
  //for example the size of the picture has been set to a 1000 x 500
  //resolution.
  AdvChartView1.Panes[0].Chart.SaveToImage(SavePictureDialog1.FileName,
  1000, 500, itPNG);
end;
```

Result when choosing 0 and 100 %.

0%



100 %



TMS VCL Chart Tips and FAQ

Programmatically scrolling and zooming in the chart

Scrolling and zooming via code in the chart is simple. This is just a matter of programmatically changing the properties Pane.Range.RangeFrom and Pane.Range.RangeTo. To illustrate this, we create a single sinus series in a chart and add 4 buttons to control the scrolling & zooming:

Initialization of the chart:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: integer;
begin
  advchartview1.BeginUpdate;
  advchartview1.Panes.Clear;
  advchartview1.Panes.Add;
  advchartview1.Panes[0].Options := advchartview1.Panes[0].Options +
[poMoving, poHorzScroll, poHorzScale];
  advchartview1.Panes[0].Series.Add;

  with advchartview1.Panes[0].Series[0] do
  begin
    for i := 1 to 200 do
      AddSinglePoint(50 * sin(i/20*PI));
  end;
  advchartview1.Panes[0].Series[0].AutoRange := arCommon;
  advchartview1.Panes[0].Range.RangeFrom := 0;
  advchartview1.Panes[0].Range.RangeTo := 200;

  advchartview1.EndUpdate;
end;

Code to perform zoom and scroll:
procedure TForm1.Scroll(delta: integer);
begin
  advchartview1.BeginUpdate;
  advchartview1.Panes[0].Range.RangeFrom :=
advchartview1.Panes[0].Range.RangeFrom + delta;
  advchartview1.Panes[0].Range.RangeTo :=
advchartview1.Panes[0].Range.RangeTo + delta;
  advchartview1.EndUpdate;
end;

procedure TForm1.Zoom(delta: integer);
begin
  advchartview1.BeginUpdate;
  advchartview1.Panes[0].Range.RangeFrom :=
advchartview1.Panes[0].Range.RangeFrom + delta;
  advchartview1.Panes[0].Range.RangeTo :=
advchartview1.Panes[0].Range.RangeTo - delta;
  advchartview1.EndUpdate;
end;

The code to zoom and scroll can then simply be attached to buttons:
```

```
procedure TForm1.ScrollLeftClick(Sender: TObject);
begin
  Scroll(-10);
end;

procedure TForm1.ScrollRightClick(Sender: TObject);
begin
  Scroll(+10);
end;

procedure TForm1.ZoomOutClick(Sender: TObject);
begin
  Zoom(+10);
end;

procedure TForm1.ZoomInClick(Sender: TObject);
begin
  Zoom(-10);
end;
```

Using Y-axis values per series in the chart

Each series in the chart can have its own Y-axis values. It can be chosen if the Y-axis values for a series should be displayed left or right from the chart. The settings to control this are under Serie.YAxis.Position. When two series have an Y-axis displayed on the same size of the chart, the spacing between the Y-axis itself and value is controlled by Serie.YAxis.MajorUnitSpacing := 30;

In the code sample below, 4 series are added to the chart. Series 1 and 3 have the Y-axis on the left side while series 2 and 4 have the Y-axis on the right side. To make clear what Y-axis belongs to what series, the series line color is set to the same color as the Y-axis value font.

```
procedure TForm2.InitChart;
begin
  advchartview1.BeginUpdate;
  advchartview1.Panes[0].Series.Clear;

  with advchartview1.Panes[0].Series.Add do
  begin
    YAxis.Position := yLeft;
    YAxis.MajorUnitSpacing := 30;
    YAxis.MajorFont.Color := clRed;
    AddSinglePoint(0);
    AddSinglePoint(15);
    AddSinglePoint(17);
    AddSinglePoint(18);
    LineColor := clRed;
    AutoRange := arEnabled;
  end;

  with advchartview1.Panes[0].Series.Add do
  begin
    YAxis.Position := yRight;
    YAxis.MajorUnitSpacing := 30;
    YAxis.MajorFont.Color := clGreen;
    AddSinglePoint(150);
    AddSinglePoint(100);
    AddSinglePoint(120);
    AddSinglePoint(190);
```

```
    LineColor := clGreen;
    AutoRange := arEnabled;
  end;

  with advchartview1.Panes[0].Series.Add do
  begin
    YAxis.Position := yLeft;
    YAxis.MajorFont.Color := clBlue;
    AddSinglePoint(20);
    AddSinglePoint(25);
    AddSinglePoint(27);
    AddSinglePoint(28);
    LineColor := clBlue;
    AutoRange := arEnabled;
  end;

  with advchartview1.Panes[0].Series.Add do
  begin
    YAxis.Position := yRight;
    YAxis.MajorFont.Color := clYellow;
    AddSinglePoint(250);
    AddSinglePoint(200);
    AddSinglePoint(210);
    AddSinglePoint(380);
    LineColor := clYellow;
    AutoRange := arEnabled;
  end;

  advchartview1.Panes[0].YAxis.Position := yBoth;
  advchartview1.Panes[0].YAxis.Size := 150;
  advchartview1.Panes[0].Range.RangeFrom := 0;
  advchartview1.Panes[0].Range.RangeTo := 3;
  advchartview1.EndUpdate;
end;
```

Add custom X-axis text programmatically

With the latest version of the TAdvChartView component it is possible to add custom X-axis text by a new override of the AddSinglePoint method that has a parameter XAxisText.
The sample code snippet here shows how to add a custom numbering as X-axis label text rotated by 40°.

```
procedure TForm.FormCreate(Sender: TObject);
var
  i: integer;
  xval: Double;
begin
  with AdvChartView1.Panes[0] do
  begin
    //Set range
    Range.RangeFrom := 0;
    Range.RangeTo := 100;
    Range.MaximumScrollRange := 100;
    Range.MinimumScrollRange := 0;
    //Set X-axis size
    XAxis.Size := 60;
    //Add points with custom X-axis text
    with Series[0] do
```

```
    begin
      ChartType := ctBar;
      AutoRange := arEnabledZeroBased;
      Color := clBlue;
      ColorTo := clSilver;
      for I := 0 to 100 do
      begin
        xval := 144 + (I * 0.2);
        AddSinglePoint(RandomRange(20, 100),  FloatToStr(xval));
      end;

      //Rotate text
      XAxis.TextBottom.Angle := 40;
      //Enabled tickmarks
      XAxis.TickMarkColor := clBlack;
    end;
  end;
```

Add custom X-axis values via an event

To override the standard X-axis values drawn by TAdvChartView an event OnXAxisDrawValue event is available for each series. To add an event handler for OnXAxisDrawValue, declare a procedure and assign it to the serie.OnXAxisDrawValue event like in the code snippet below:

```
AdvChartView.Panes[0].Series[0].OnXAxisDrawValue := XAxisDrawValue;

procedure TForm.XAxisDrawValue(Sender: TObject; Serie: TChartSerie;
  Canvas: TCanvas; ARect: TRect; ValueIndex, XMarker: integer; Top:
Boolean;
  var defaultdraw: Boolean);
begin
  //Insert Code here
end;
```

In this event, the Serie parameter returns the Series for which the X-axis value needs to be drawn. Further, the Canvas and rectangle within this canvas where can be drawn is returned. ValueIndex returns the index of the value along the X-axis. Top indicates whether the X-axis is on top or below the chart. Finally, by setting the DefaultDraw parameter to false, the chart itself will no longer draw the X-axis value.

The technique illustrated with following sample. Some points were added in a series and for each odd value on the X-axis we want to display a Car brand in the X-axis:

```
procedure TForm.XAxisDrawValue(Sender: TObject; Serie: TChartSerie;
  Canvas: TCanvas; ARect: TRect; ValueIndex, XMarker: integer; Top:
Boolean;
  var defaultdraw: Boolean);

const
  lst: array[0..4] of String = ('Audi', 'BMW', 'Mercedes', 'Bugatti',
'Porsche');
var
  s: String;
  th, tw: integer;
begin
  if Odd(ValueIndex) then
  begin
    Canvas.Font.Size := 12;
```

```
    s := lst[Random(Length(lst))];
    th := Canvas.TextHeight(s);
    tw := Canvas.TextWidth(s);
    Canvas.TextOut(Xmarker - (tw div 2), ARect.Top + th, s);
  end;
end;
```

Updating chart at runtime

When updating the chart component at runtime it is important to "tell" the chart to update itself.

Whenever you change properties, or add/remove points the chart will not be updated.
This is because the chart will be repainted over and over again every time you change a property.
This would cause a slow performance.

To update the chart at runtime, after you change properties, you must use the BeginUpdate / EndUpdate methods.

Sample: Updating a series of points at runtime

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  AdvChartView1.BeginUpdate;
  with AdvChartView1.Panes[0].Series[0] do
  begin
    Points[0].SingleValue := 100;
    Points[1].SingleValue := 50;
    Points[2].SingleValue := 20;
    Points[3].SingleValue := 30;
    Points[4].SingleValue := 150;
    Points[5].SingleValue := 160;
  end;
  AdvChartView1.EndUpdate;
end;
```

Modifying a single series data point value in code

If a data value changes after it was added to the series, it is easy to update the value at a later time via code. To do this, it is required to know the index of the pane, the index of the series and the index of the data point in the series. This example code snippet changes the third datapoint of first series in the first pane of the chart:

```
AdvChartView1.BeginUpdate;
AdvChartView1.Panes[0].Series[0].Points[2].SingleValue := Random(100);
AdvChartView1.EndUpdate;
```

Retrieving the series values at crosshair

While the tracker window will display series values at crosshair position, it can often be interesting to retrieve the values in code. Following code snippet shows the series values in the caption of the form:

```
procedure TForm1.AdvChartView1MouseMove(Sender: TObject; Shift: TShiftState
;
  X, Y: Integer);
var
  i: integer;
  cp: TChartPoint;
  s: string;
begin
  with AdvChartView1.Panes[0] do
  begin
    s := '';
    for i := 0 to Series.Count - 1 do
    begin
      cp := GetChartPointAtCrossHair(i);
      if s = '' then
        s := floattostr(cp.SingleValue)
      else
        s := s + ':' + floattostr(cp.SingleValue);
    end;
  end;
end;
```

TMS VCL Chart 3D use

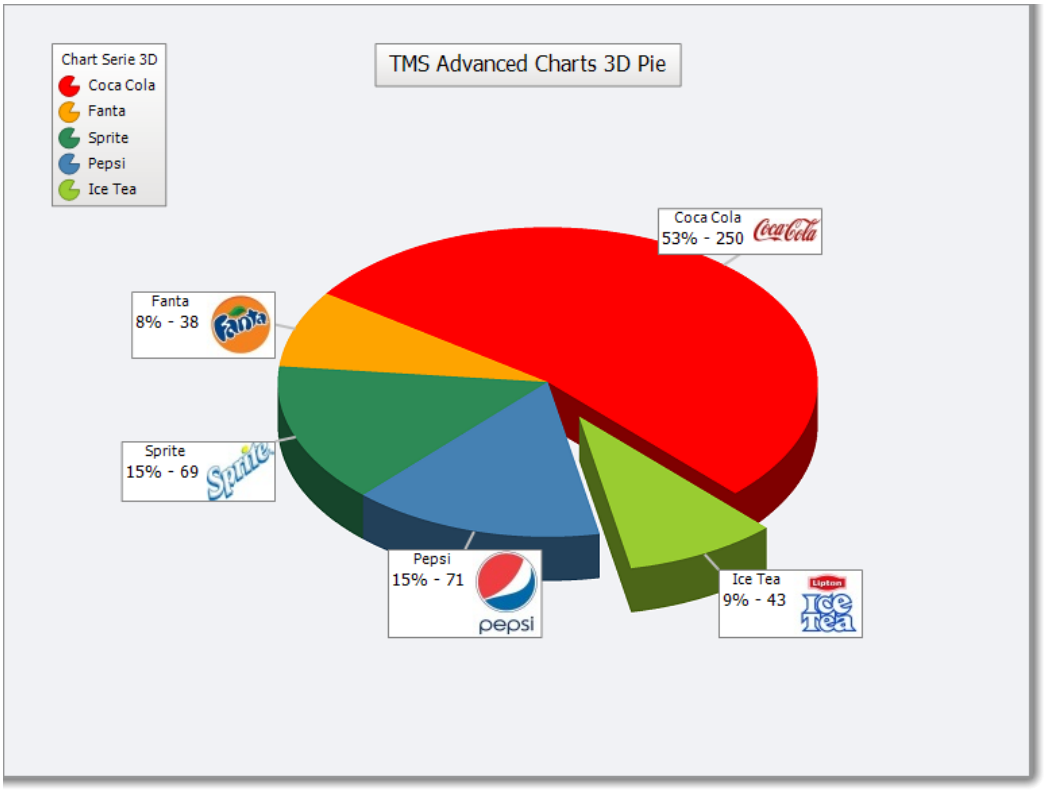
The TMS VCL Chart 3D is designed to display series rendered in 3D via OpenGL.

TMS VCL Chart 3D organisation

TMS VCL Chart 3D includes editors to edit series and points that can be used at design-time and run-time.

The visual organisation of TMS VCL Chart 3D

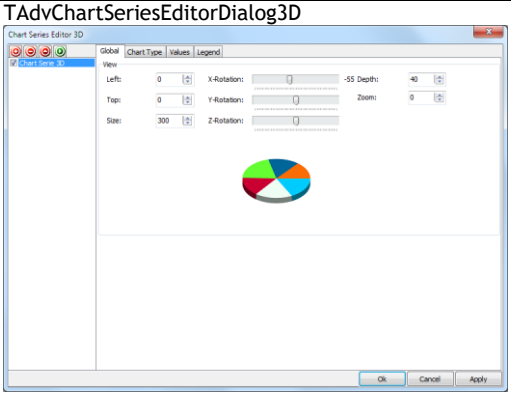
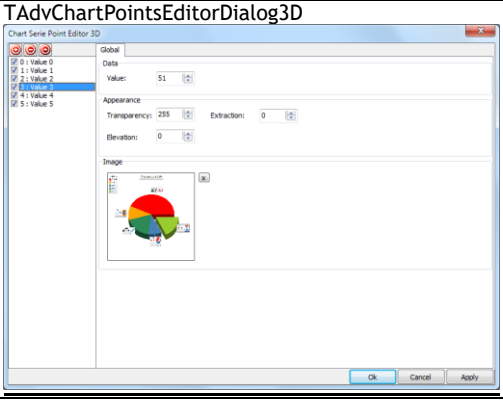
TAdvChartView3D



1) Chosen chart type displayed in precalculated area (depending on the visibility and the amount of series). The chart is rendered in OpenGL and has full 3D capabilities such as X, Y and Z rotation.
2) Values displayed on the chart. Values can contain a caption, value and percentage calculated value, as well as an optional image drawn with aspect ratio taken in account.
3) Legend showing color and the caption of each series and listing the visible values inside the chart.
4) Title showing the caption of the ChartView3D control.

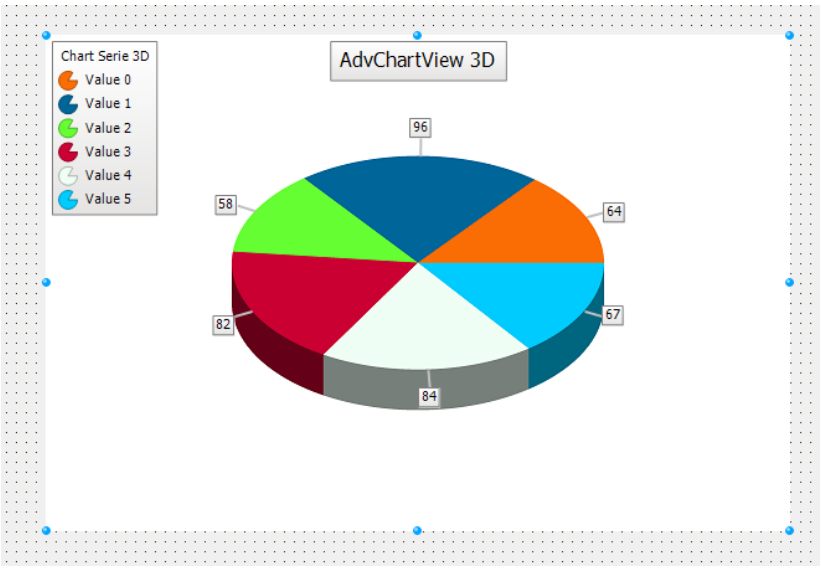
TAdvChartSeriesEditorDialog3D, TAdvChartPointsEditorDialog3D

All properties of the TAdvChartView3D component can be edited at runtime via components in the same way as at design-time. Therefore two editors are installed in the component palette to easily change properties of the series and series point collection. In each separate dialog you can remove existing or add new elements and simply cancel or save your changes.

TAdvChartSeriesEditorDialog3D 	The Series editor dialog is used to change the appearance / layout of the series, series legend and series values. The Series editor is called at design-time by double-clicking on the chart.
TAdvChartPointsEditorDialog3D 	The Points editor dialog is a smaller editor which only edits the Points attached to the chosen series. The Points editor is called by double-clicking on the name of the series in the series editor or by clicking on the green button in the toolbar.

TMS VCL Chart 3D in detail

When dropping a new TAdvChartView3D component, the component is initialized with a series and a few default series points.



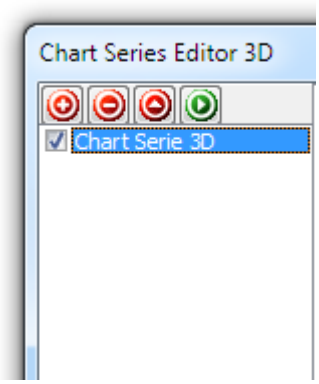
The equivalent in code is `AdvChartView3D.InitSample`.

Title

The chart title has a background fill and border and can be configured with a text, font and position property.

Series

The chart has a series collection that can be modified at designtime and at runtime. At designtime, the series can be added, removed / renamed by double-clicking on the chart, and displaying the series editor dialog.

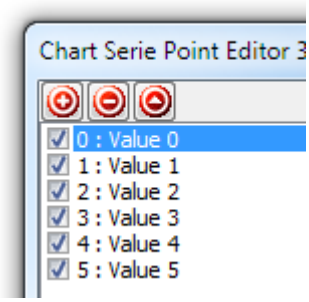


At runtime, the series can be added / modified by encapsulating the code with a BeginUpdate and EndUpdate. This is necessary for all properties or methods called or set at runtime. A BeginUpdate / EndUpdate combination gives you are better performance in updating the chart. Below is a sample code to add / remove a series at runtime.

```
AdvChartView3D1.BeginUpdate;
AdvChartView3D1.Series.Add;
AdvChartView3D1.EndUpdate;

AdvChartView3D1.BeginUpdate;
AdvChartView3D1.Series.Delete(0);
AdvChartView3D1.EndUpdate;
```

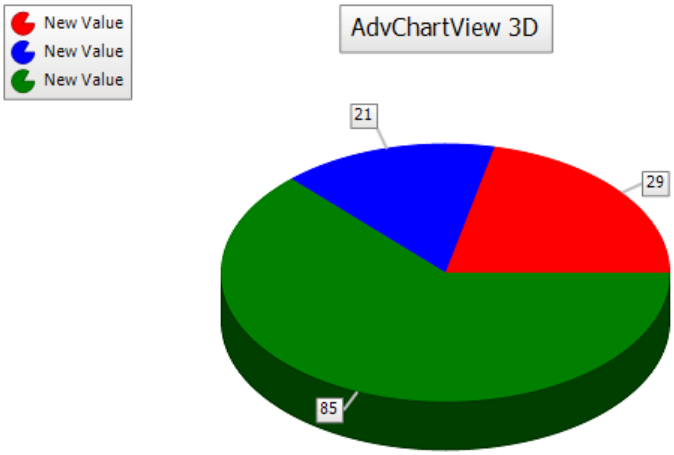
Points

After adding a series, adding points to a series is done in a similar way, through an items collection. Points (Items) can be added to the series at design-time and at run-time. At design-time, the points can be added, removed / renamed by double-clicking on the series of choice in the series editor dialog.



At run-time, points can be added / modified by first accessing the series and adding or modifying a point from the items collection. Below is a sample that adds a new series on an empty chart, and adds 3 random values with different colors.

```
var
  cs: TChartSerie3D;
  ci: TChartItem3D;
begin
  AdvChartView3D1.BeginUpdate;
  AdvChartView3D1.Series.Clear;
  cs := AdvChartView3D1.Series.Add;
  ci := cs.Items.Add;
  ci.Color := clRed;
  ci.Value := Random(100);

  ci := cs.Items.Add;
  ci.Color := clBlue;
  ci.Value := Random(100);

  ci := cs.Items.Add;
  ci.Color := clGreen;
  ci.Value := Random(100);

  AdvChartView3D1.EndUpdate;
end;
```



Legend

For each series there is a legend, the legend is positioned top left by default and can be modified in the series editor dialog. The legend can display a caption, and the value captions added to the series.

Values

The values of the series points are displayed on the chart by default. In the series editor dialog, this can be modified to show additional information such as the value caption, the percentage of the value related to the total value of all points and an optional image.

Methods

Chart

```
procedure SaveToImage(Filename: String; ImageWidth, ImageHeight: integer;
ImageType: TImageType = itBMP; ImageQualityPercentage: integer = 100);
```

Saves the chart to an image file. Supported file formats: BMP,PNG,JPEG,GIF.

```
function XYToSerie(X, Y: Integer): TChartSerie3D;
```

Returns the series at an X and Y position.

```
procedure UpdateChart;
```

Forces an update of the chart.

```
procedure InitSample;
```

Initializes a sample of the chart.

```
procedure BeginUpdate;
```

Blocks all run-time updates. Needs to be used in combination with EndUpdate.

```
procedure EndUpdate;
```

Ends blocking all run-time updates and automatically updates the chart. Needs to be used in combination with BeginUpdate.

Properties

Chart

```
property Color: TColor
```

The color used for the background of the chart.

```
property Series: TChartSeries3D
```

The collection of series inside the chart. TChartView3D allows multiple series to be displayed, horizontally divided in the control.

```
property Title: TChartTitle3D
```

The title in the chart.

Chart - Title

```
property Alignment: TChartTextAlignment3D
```

The alignment of the title text, the title text can be positioned left, center or right.

```
property Border: TChartLine3D
```

The border of the background of the title.

```
property Fill: TChartGradientFill3D
```

The fill of the background of the title.

```
property Font: TFont
```

The font of the title.

```
property Margin: integer
```

The margin of the title text between the top, left, bottom and right of the title area.

```
property Padding: integer
```

The padding of the title between the top, left, bottom and right of title area and the chart area.

```
property Position: TChartItemPosition3D
```

The position of the title, the title can be positioned top, left, bottom, right, center or a combination of these positions.

```
property Transparency: Integer
```

The transparency of the title fill and text.

```
property Text: string
```

The text of the title.

```
property Visible: Boolean
```

Shows or hides the title.

Chart - Series

```
property Caption: string
```

The caption of the series, used to identify the series and used inside the legend as caption.

```
property ChartType: TChartType3D
```

The chart type of the series.

```
property Depth: single
```

The depth of the 3D view of the series.

```
property Interaction: Boolean
```

Enables mouse left and right button and mouse wheel interaction.

```
property Items: TChartItems3D
```

The Items / Points of the series.

```
property Left: single
```

The left position of the series within the control's client area.

```
property Legend: TChartLegend3D
```

The legend of the series.

```
property Size: single
```

The size of the series.

```
property SizeType: TChartSizeType3D
```

The size type of the series. The size type can be set to percentage, which, in combination with the size property is used to render the series on a percentage of the chart area. The size type can also be set to pixels which will use the actual size value in pixels to render the series.

```
property Top: single
```

The top position of the series.

```
property Values: TChartValues3D
```

The values displayed on the series.

```
property Visible: Boolean
```

Shows / hides the series.

```
property XRotation: single
```

The X-Rotation of the 3D view of the series.

```
property YRotation: single
```

The Y-Rotation of the 3D view of the series.

```
property ZRotation: single
```

The Z-Rotation of the 3D view of the series.

Chart - Series - Items/Points

```
property Caption: String
```

The caption of the item / point that is used inside the legend and inside the values area.

```
property Color: TColor
```

The color of the item / point, which is used to for the rendering.

```
property Elevation: Single
```

The elevation of an item / point, the rendering will elevate the element specifically for that item. The elevation is performed in the Y-direction.

```
property Expansion: Single
```

The expansion of an item / point, the rendering will expand the element specifically for that item. The expansion is performed in the Y-direction.

```
property Extraction: Single
```

The extraction of an item / point, the rendering will extract the element specifically for that item. The extraction is performed in the X-direction.

```
property Image: TChartGDIPPicture
```

The image used in the value area if ImageVisible property on series level is true.

```
property Value: Extended
```

The Value of the item / point used to calculate the series elements that are rendered in the chart.

```
property Visible: Boolean
```

Shows / hides the item / point. When Visible is false, the item is not taken into account for percentage calculation for example.

```
property Transparency: Integer
```

The transparency of the item / point.

Chart - Series - Values

```
property Border: TChartLine3D
```

The border of the value area.

```
property CaptionAlignment: TChartTextAlignment3D
```

The alignment of the captions displayed in the value area.

```
property CaptionsFont: TFont
```

The font of the captions displayed in the value area.

```
property Fill: TChartGradientFill3D
```

The fill of the value area.

```
property ImageVisible: Boolean
```

If a point image is assigned, the image is displayed inside the value area if this property is true.

```
property ImageAspectRatio: Boolean
```

Enables or disables aspect ratio on the image.

```
property ImageHeight: Integer
```

The height of the image when ImageAspectRatio is false.

```
property ImageWidth: Integer
```

The width of the image when ImageAspectRatio is false.

```
property ImagePosition: TChartItemPosition3D
```

The position of the image inside the value area.

```
property ShowCaptions: Boolean
```

Shows the captions of the points in the value area.

```
property ShowValues: Boolean
```

Shows the values of the points in the value area formatted with ValueFormat.

```
property ShowPercentages: Boolean
```

Shows the values in percentage of the points in the value area.

```
property Transparency: Integer
```

Transparency of the captions, values and value area.

```
property TickMarkLength: Integer
```

The length of the tickmark used to identify a value.

```
property TickMarkSize: Integer
```

The size of the tickmark used to identify a value.

```
property TickMarkColor: TColor
```

The color of the tickmark used to identify a value.

```
property ValuesFont: TFont
```

The font of the values displayed in the value area.

```
property ValuesAlignment: TChartTextAlignment3D
```

The alignment of the values displayed in the value area.

```
property ValueFormat: String
```

The format of the values displayed in the value area.

```
property Visible: Boolean
```

Enables / Disables values.

Chart - Series - Legend

```
property Border: TChartLine3D
```

The border of the legend.

```
property CaptionAlignment: TChartTextAlignment3D
```

The caption alignment of the legend.

```
property CaptionFont: TFont
```

The font of the caption of the legend.

```
property CaptionVisible: Boolean
```

Shows / hides the caption of the legend.

```
property Fill: TChartGradientFill3D
```

The fill of the legend.

```
property ItemsFont: TFont
```

The font of the items / values of the legend.

```
property Margin: integer
```

The margin of the legend caption and values between the top, left, bottom and right of the legend area.

```
property Padding: integer
```

The padding of the legend between the top, left, bottom and right of title area and the chart area.

```
property Position: TChartItemPosition3D
```

The position of the legend, the legend can be positioned top, left, bottom, right, center or a combination of these positions.

```
property Transparency: Integer
```

The transparency of the caption, values and legend area.

```
property Visible: Boolean
```

Shows / hides the legend.

Fill

```
property Direction: TChartGradientDirection3D
```

The direction of the gradient. The direction can be set to vertical or horizontal.

```
property Color: TColor
```

The start color of the gradient.

```
property EndColor: TColor
```

The end color of the gradient.

```
property Visible: Boolean
```

Shows / hides the gradient.

Border

```
property Color: TColor
```

The color of the border.

```
property Style: DashStyle
```

The style of the border.

```
property Visible: Boolean
```

Shows / hides the border.

```
property Width: Integer
```

The width of the border.

Multiple Series

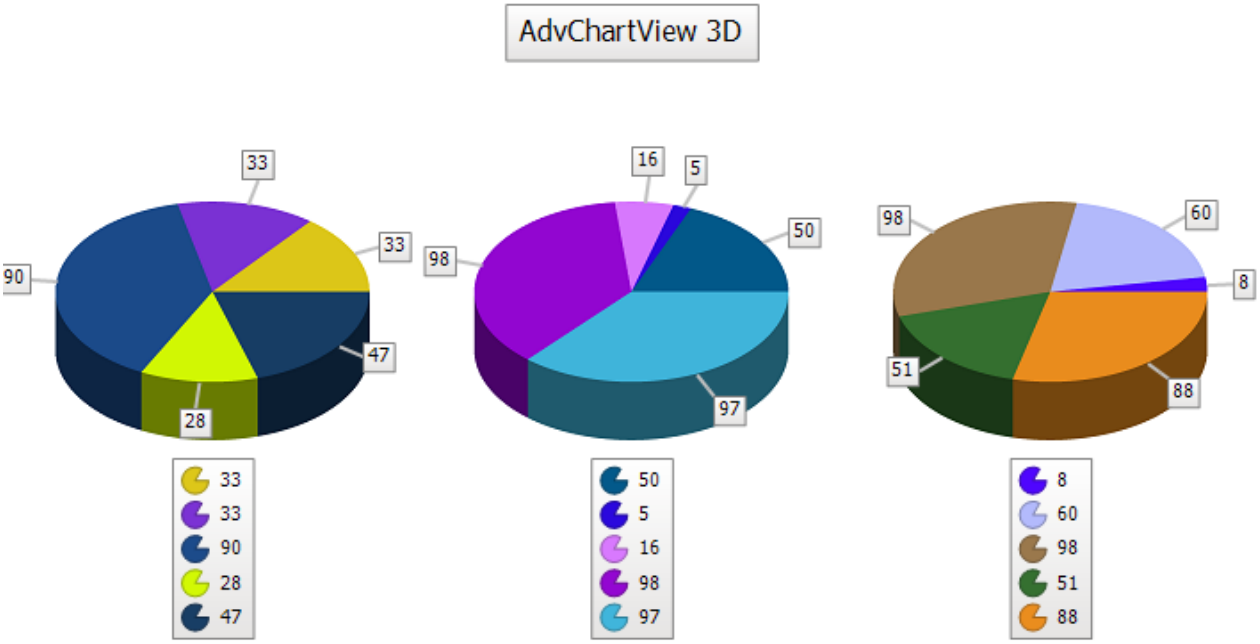
The chart supports multiple series. When dropping a chart on the form, the chart already contains 1 series. Adding / removing series can be done by double-clicking on the chart component to start the series editor dialog.

When multiple series have been added, the chart automatically divides the area width by the amount of series. All series are positioned from left to right starting with the first serie in the collection. Each legend is positioned in the preserved area which has been calculated based on the amount of series so positioning a legend is done relative to the series area.

Below is a sample with multiple series:

```
var
  I: Integer;
  K: Integer;
  cs: TChartSerie3D;
  ci: TChartItem3D;
begin
  AdvChartView3D1.BeginUpdate;
  AdvChartView3D1.Series.Clear;
  for I := 0 to 2 do
  begin
    cs := AdvChartView3D1.Series.Add;
    cs.Legend.Position := ipBottomCenter;
```

```
    cs.Size := 75;
    cs.SizeType := stPercentage;
    for K := 0 to 4 do
    begin
      ci := cs.Items.Add;
      ci.Color := RGB(Random(255), Random(255), Random(255));
      ci.Value := Random(100);
    end;
  end;
  AdvChartView3D1.EndUpdate;
```



Interaction is supported for multiple series and is executed by clicking the series area of choice.

Interaction

By default interaction is enabled for each existing or newly created series. The Interaction property can be used to turn interaction on or off. When interaction is enabled, clicking and dragging the mouse on the area of the series will rotate the series around the X, Y or Z axis depending on the movement and mouse button.

When dropping an instance of the chart on the form, you will notice the chart is already rotated on the X-Axis. The rotation values go from -180 to 180, but the properties are not limited to these values and accept smaller and larger values. Internally these values will be converted to an equivalent used for rendering. Below are some samples of rotation around the X, Y and Z-Axis.

X-Rotation = -50 Y-Rotation = 0 Z-Rotation = 0	X-Rotation = 0 Y-Rotation = -50 Z-Rotation = 0	X-Rotation = 0 Y-Rotation = 0 Z-Rotation = -50

To rotate around the X-Axis with the mouse, click and hold the left mouse button on the serie of choice and move the mouse from left to right or vice versa.

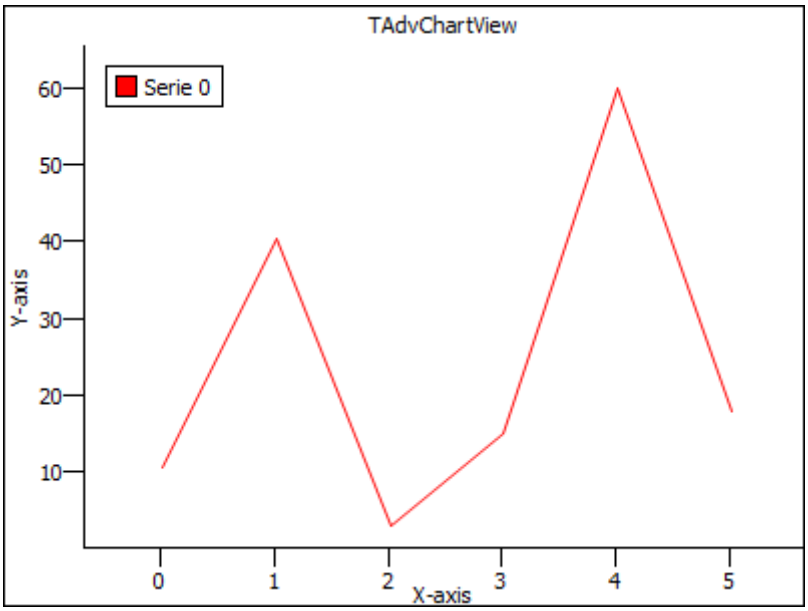
To rotate around the Y-Axis with the mouse, click and hold the left mouse button on the serie of choice and move the mouse from top to bottom or vice versa.
To rotate around the Z-Axis with the mouse, click and hold the right mouse button on the serie of choice and move the mouse from left to right or vice versa.

With interaction enabled zooming the chart can be done by scrolling the mouse wheel.

Virtual mode

The virtual mode is enabled as soon as you implement the OnGetNumberOfPoints. After implementing the OnGetNumberOfPoints, the OnGetPoint event is called to retrieve the data for a point. This is done through a record that can be directly accessed and manipulated. The advantage is that the event signature will not change when adding more properties in the future. Below is a sample that demonstrates this.

```
const
  PointArray: array[0..10] of Double = (10.5, 40.4, 3, 15, 60, 18, 34,
40.5, 15.9, 35, 4);

procedure TForm111.AdvGDIPChartView1GetNumberOfPoints(Sender: TObject;
Pane,
  Serie: Integer; var ANumberOfPoints: Integer);
begin
  ANumberOfPoints := Length(PointArray);
end;

procedure TForm111.AdvGDIPChartView1GetPoint(Sender: TObject; Pane, Serie,
  AIndex: Integer; var APoint: TChartPoint);
begin
  APoint.SingleValue := PointArray[AIndex];
end;

procedure TForm111.FormCreate(Sender: TObject);
begin
  AdvGDIPChartView1.BeginUpdate;
  AdvGDIPChartView1.Panes[0].Series.Clear;
  AdvGDIPChartView1.Panes[0].Series.Add;
  AdvGDIPChartView1.EndUpdate;
end;
```



AntiAliasing

The chart renders the 3D view with AntiAliasing enabled by default. The performance when resizing the form or interaction with the view is more CPU intensive when anti-aliasing is turned on. Therefore the chart exposes a public property to switch off antialiasing to provide faster interaction / animation capabilities.

```
AdvChartView3D1.AntiAlias := True;
```

Provides high quality rendering, low performance.

```
AdvChartView3D1.AntiAlias := False;
```

Provides high performance, low quality rendering.