

# Tutorial



*X2max*  
www.x2max.com

FileMaker Pro is a registered trademark of FileMaker Inc.  
©1997-2012 by X2max Software. All rights reserved.

# Table Of Contents

## Getting Started

<b>Requirements</b>	<b>8</b>
Operating System	8
FileMaker Pro Version	8
Knowledge of FileMaker Pro	8
<b>Installation</b>	<b>9</b>
Installation	9
Support	9
<b>Basics</b>	<b>10</b>
What is xmCHART	10
How does xmCHART work	10
A basic example	12

## Components

<b>Overview</b>	<b>17</b>
<b>Layout</b>	<b>18</b>
OpenDrawing()	18
CloseDrawing()	20
OpenView()	20
CloseView()	21
OpenChart()	21
CloseChart()	25
<b>Data</b>	<b>26</b>
ChartData()	26

ChartDataOptions()	27
ChartDataUpperLimits()	29
ChartDataLowerLimits()	29
ChartDataRead()	31
ChartDataWrite()	32
Entry of Date and Time	34
<b>Charts</b>	<b>36</b>
ScatterChart()	36
ScatterChart2D()	39
LineChart()	40
LineChart2D()	44
AreaChart()	46
AreaChartOptions()	50
AreaChart2D()	53
BarChart()	57
BarChart2D()	67
BarChartOptions()	67
GanttChart()	70
BubbleChart()	75
BubbleChartOptions()	78
BubbleChart 2D()	78
PieChart()	80
PieChartExplodes()	85
PieChartExplodeDepths()	87
PieChartAuxLines()	88
PieChartLabelOptions()	89
PieChartInnerLabelTexts()	90
PieChartInnerLabelStyle()	90
PieChartInnerLabelBackground()	90
PieChartCenterLabelText()	91
PieChartCenterLabelStyle()	91
PieChartCenterLabelBackground()	91
PolarChart()	92
PolarChartOptions()	96
RadarChart()	100
RadarChartOptions()	104
HighLowChart()	108
HighLowChart2D()	115
CandlestickChart()	116
CandlestickChart2D()	119
Histogram()	120
HistogramRange()	120
HistogramOptions()	123
BoxPlot()	128

BoxPlotOptions()	129
<b>Styles</b>	<b>136</b>
FillStyle()	136
PictureStyle()	138
BorderStyle()	140
LineStyle()	142
SymbolStyle()	144
ShadowStyle()	146
ArrowStyle()	148
LabelTexts()	150
LabelStyle()	153
LabelBackground()	158
LabelOptions()	159
<b>Background</b>	<b>171</b>
Background()	171
BackgroundPict()	175
ChartBackground()	179
ChartBackgroundPict()	181
<b>Grids</b>	<b>184</b>
MajorGridLineWidths()	184
MajorGridLineColors()	184
MajorGridLinePatterns()	184
MinorGridLineWidths()	186
MinorGridLineColors()	186
MinorGridLinePatterns()	186
MajorGridStripeColors()	189
MajorGridStripePatterns()	189
MinorGridStripeColors()	189
MinorGridStripePatterns()	189
GridFrame()	192
GridLocation()	193
<b>Axes</b>	<b>195</b>
Scaling()	195
ScalingOptions()	204
AxisLine()	207
AxisMajorTicks()	207
AxisMinorTicks()	207
AxisOptions()	209
AxisLabelText()	211
AxisLabelStyle()	214
AxisLabelBackground()	215

AxisLabelOptions()	217
AxisMajorTickLabelTexts()	219
AxisMajorTickLabelStyle()	221
AxisMajorTickLabelBackground()	222
AxisMajorTickLabelOptions()	223
AxisMinorTickLabelTexts()	226
AxisMinorTickLabelStyle()	226
AxisMinorTickLabelBackground()	226
AxisMinorTickLabelOptions()	226
<b>Legend</b>	<b>230</b>
LegendTexts()	230
LegendStyle()	231
LegendBackground()	232
LegendOptions()	233
<b>Title</b>	<b>241</b>
TitleText()	241
TitleStyle()	242
TitleSubStyle()	242
TitleBackground()	243
TitleOptions()	244
<b>Drop Lines</b>	<b>247</b>
DropLineStyle()	247
DropLineReferencePoint()	249
DropLineReferenceLine()	250
DropLineReferenceSeries()	252
<b>Moving Averages</b>	<b>254</b>
MovingAverage()	254
MovingAverageLineStyle()	256
MovingAverageOptions()	258
<b>Curve Fitting</b>	<b>267</b>
CurveFitting()	268
CurveFittingLineStyle()	270
CurveFittingOptions()	271
<b>Error Bars</b>	<b>276</b>
ErrorBars()	276
ErrorBarData()	278
ErrorBarStyle()	279
ErrorBarStyle2D()	281

<b>Graphic Primitives</b>	<b>283</b>
AddText()	284
AddLine()	287
AddFrame()	288
AddRect()	289
AddRoundFrame()	290
AddRoundRect()	291
AddEllipse()	292
AddOval()	293
AddArc()	294
AddSlice()	295
AddPolyline()	296
AddPolygon()	297
AddSmoothPolyline()	298
AddSmoothPolygon()	299
AddPath()	300
AddSymbol()	304
AddArrow()	306
AddPicture()	307
AddClipRect()	308
AddClipRoundRect()	308
AddClipOval()	308
AddClipSlice()	308
AddClipPolygon()	308
AddClipSmoothPolygon()	308
AddClipReset()	309
<b>Output</b>	<b>310</b>
SendToClipboard()	310
SaveAsPDFFile()	311
SaveAsEMFFile()	312
SaveAsPICTFile()	312
SaveAsGIFFile()	313
SaveAsJPGFile()	313
SaveAsPNGFile()	313
SaveAsBMPFile()	314
SaveAsSVGFile()	314
SaveAsTIFFFile()	314
<b>Miscellaneous</b>	<b>315</b>
DateTimeOptions()	315
SetDecimalPoint()	316
SetThousandsSep()	316
<b>Index</b>	<b>318</b>

# Getting Started

# Requirements

**Operating System:**

- Windows XP, Vista, Windows 7
- Mac OS X 10.5 or higher

**FileMaker Pro Version:**

Version: 8/8.5/9/10/11/12

FileMaker Pro 7.0 is supported up to xmCHART v3.1

(For FileMaker Pro 4/5/5.5/6 xmCHART v2.2 is required.)

**Knowledge of FileMaker Pro:**

In order to be able to work with xmCHART successfully, the following knowledge of FileMaker Pro is required:

- Defining FileMaker Pro fields, particularly global fields.
- Designing and revising layouts.
- Working with so-called *external functions*. Plug-ins are linked with FileMaker Pro databases via external functions.
- Basic knowledge of FileMaker Pro scripting. Charts can be created automatically by using FileMaker Pro scripts.

The FileMaker Pro example databases, that come with *xmExamples*, provide additional ideas, comments and support.



# Installation

**Installation:**

- Quit FileMaker Pro if necessary.
- Copy xmCHART (plug-in) into the "Extensions" folder which is normally located in the "FileMaker Pro" folder.
- Start FileMaker Pro.

Furthermore, you must make sure that xmCHART is activated. This can be checked in the "*Preferences...>Plug-Ins*" menu.

xmCHART can also be easily linked with FileMaker Pro Runtime solutions. For this purpose, make a folder named "Extensions" within the folder which contains the FileMaker Pro files for your runtime solution. Then copy xmCHART (plug-in) into this folder.

**Support:**

Should problems occur during installation, please send an e-mail containing the following information:

- Platform and operating system version
- FileMaker Pro version
- A description of the error

to: <[support@x2max.com](mailto:support@x2max.com)>. We will make every effort to get in touch with you as soon as possible and find a solution.

# Basics

## What is xmCHART

xmCHART is a very powerful FileMaker Pro plug-in for creating all important types of charts such as bar, scatter, line, area, bubble, high-Low, candlestick, pie, radar, polar and Gantt charts. By means of an extensive set of functions, all chart components such as scalings, axes, grids and labels can be controlled by the user. Furthermore, a title, legend, background picture, additional texts and picture information, e.g. a logo, can be easily added to complete a chart.

Therefore, xmCHART opens up a wealth of new ways for analyzing and representing numerical data for both FileMaker Pro developers and users.

## How does xmCHART work

The basic idea behind xmCHART is to make an extensive set of functions available to users with which the general appearance of a chart can be controlled in detail. Three functions are all you need to draw up a number of different charts: first the function `OpenDrawing()` defines the size of the chart, secondly the `ChartData()` function which contains all numerical values used for representation and, thirdly, a function which defines the type of chart, for example:

```
OpenDrawing(400;300) // width & height
ChartData(23 45 -2.76 12.9 5; 24 13 8.5 -3)
BarChart(shadow+horizontal)
```

In fact, over 150 functions are presently available with which nearly all parts of the chart can be varied. All function calls are entered in one single FileMaker Pro text field line by line – in other words, one function per line – either manually or automatically by means of a FileMaker Pro script. The advantages of working in this manner are:

- **Clarity**

All function calls necessary for drawing a chart are clear, combined into one single text field and, in addition, can be supplemented with explanatory comments.

- **Easy Developing and Testing**

Modifications can be tested immediately — without accessing the Script Editor. Possible errors are precisely specified by giving the line number, function name and argument index.

*TIP: Functions that are not necessary can be deactivated in the developing and testing stage simply by putting 2 slashes "/" as a comment symbol in front of them or by a multi-line comment /\* ... \*/.*

- **Portability and Support**

Function calls can be moved to other FileMaker Pro databases or sent by e-mail by simply using the usual *Copy* and *Paste* editing commands, e.g. send questions for the technical support to: <[support@x2max.com](mailto:support@x2max.com)>

If a chart is to be created now, the contents of this text field, which generally consist of a number of function calls and comment lines, is transferred to xmCHART by means of a short FileMaker Pro script. xmCHART then checks this text string for possible typing errors, missing brackets, invalid values, etc. In the event of an error, the process is aborted and an error message is sent to FileMaker Pro by xmCHART. To enable the user to quickly and easily correct this error, xmCHART sends a detailed error message with the line number and, if possible, the function name and argument index. If no errors are found, xmCHART begins to set up the chart; this occurs in the background without the user seeing it. As a rule, the sequence of the function calls does not affect the speed or the appearance of the chart. Errors that may possibly arise as the chart is being set up, such as the given size of chart being too small, also lead to abortion and a corresponding error message. If no errors occur, the drawing created by xmCHART is finally copied into a FileMaker container field.

The manner in which xmCHART functions can be summarized in the following three areas:

- Checking the function calls entered.
- Setting up the chart *offscreen*, that means it takes place in the background without the user seeing it occur.
- Copying the completed chart into a FileMaker container field.

## A basic example

In the following basic example five numerical values are represented graphically by a bar chart in which the bars should have labels and shadows. A title should also be added to the chart. The necessary steps can be divided into three different tasks.

### • Defining FileMaker Pro Fields

In general, xmCHART requires three FileMaker Pro fields; advantageously, they are global fields which are denoted by the prefix "g" (global) in the following.

First, a text field which contains all function calls and optional comment lines, is referred to as *gFunctions* in the following.

Secondly, a container field, which holds the chart created by xmCHART, is referred to as *gChart* in the following.

And thirdly, another global text field is called *gError*, into which possible error messages may be entered. (Fig. 1)

<u>Field Name</u>	<u>Type</u>	<u>Options</u>
gFunctions	Text	Global
gChart	Container	Global
gError	Text	Global

Fig. 1

**TIP:** *Instead of a text field and container field, a calculation field can also be used. The advantage of a calculation field is that no FileMaker Pro Script has to be called to create the chart, or, otherwise stated, the chart is automatically updated as soon as the values in the calculation field change which, in many cases, can prove to be extremely advantageous.*

- **Creating FileMaker Pro Scripts**

In general, a short FileMaker Pro script is necessary, which transfers the contents of *gFunctions* to xmCHART by means of the external function `xmCH_DrawChart()`. (see Fig. 2)

```
Set Field [ xmCHART::gChart; xmCH_DrawChart(xmCHART::gFunctions)]  
Set Field [ xmCHART::gError; xmCH_GetErrorMessage( "103" ) ]
```

Fig. 2

If the source field is a calculation field, the chart is updated automatically without launching a FileMaker script.

In the event of an error, a corresponding message is stored in *gError*; if no error occurs, *gError* is assigned an empty string.

Other scripts or *Custom Functions* are necessary when xmCHART functions are to be created dynamically; that means the contents of the text field *gFunctions* are created automatically by script commands. This is usually necessary for setting up the function `ChartData()`.

- **xmCHART Functions**

The functions needed to design and draw a chart are entered into the *gFunctions* text field. Three function calls are absolutely essential: first the function `OpenDrawing()` defines the size of the chart, secondly the `ChartData()` function which contains all numerical values used for representation and, thirdly, a function which determines the type of chart, e.g. `BarChart()`. The following types of charts are presently available: bar, scatter, line, area, bubble, high-Low, candlestick, pie, radar, polar, box&whisker plots and Gantt charts.

Please keep in mind the following rules when entering the functions: *(A detailed description can be found in the "Syntax" section of xmReference.)*

- Only one function call per line; empty lines are permitted.
- Comments are preceded by 2 slashes "//" or as multi-line comment `/* ... */`.
- If a function has several arguments, they are separated by semicolons ";"
- Optional arguments — ones that are not absolutely necessary — can be skipped when entering a function call. In this case, the default values stored in xmCHART are used. They can be taken from *xmReference*.

For example: `LegendBackground(white;;2;;3)`

- Strings and names of fonts, e.g. "Times", are to be placed in double quotes ("). If a double quote is to be issued, it must be entered *twice*.

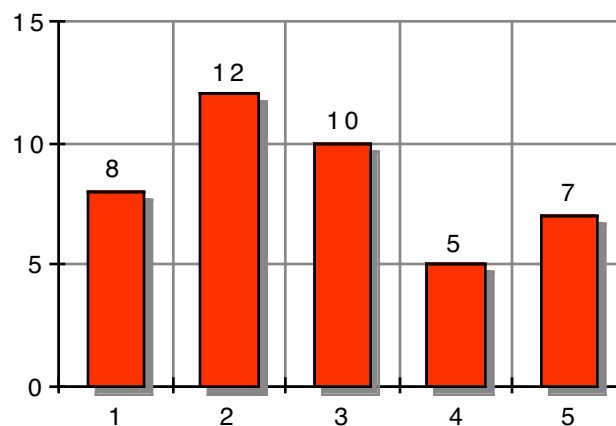
For example: `TitleText( " " "A" "BC" " " )` produces "A"BC".

PLEASE NOTE: *Double quotes (") are not to be confused with typographical (smart) quotes ("). Typographical quotes can be activated or deactivated in the "File>File Options.../Text" menu.*

The following 4 functions are needed to draw the chart described at the beginning: (see Fig. 3)

```
OpenDrawing(250;200)
ChartData(8 12 10 5 7)
BarChart(label+shadow)
TitleText("Chart 1")
```

**Chart 1**



**Fig. 3**

Please note the following:

- `OpenDrawing(width;height)`  
*width*: The entire width of the chart including axes, legend, title, etc. (in pixels)  
*height*: The entire height of the chart including axes, legend, title, etc. (in pixels)
- `ChartData()`  
The numerical values to be charted are copied to xmCHART by means of the `ChartData()` function. The individual values of a series are separated by spaces, tabs or line feeds. When transferring several series, they should be separated by semicolons. The number of values per series may vary. For example:  
`ChartData(78 -12; 45 7 -23; 0; 12 -34 78 23.5)`
- `BarChart()`  
By entering so-called *appearance constants*, the appearance of a chart can be varied. Appearance constants can be combined by using a plus sign "+". Please see *xmReference* regarding which constants are available for which type of chart.

# Components



# Components

## Overview

A chart created with xmCHART generally consists of a number of individual components such as grid, axes, legend or title. All functions provided in xmCHART together with numerous examples are described in the following sections:

- Layout
- Data
- Charts
- Styles
- Background
- Grids
- Axes
- Legend
- Title
- Drop Lines
- Moving Averages
- Curve Fitting
- Error Bars
- Graphic Primitives
- Output
- Miscellaneous

## Layout

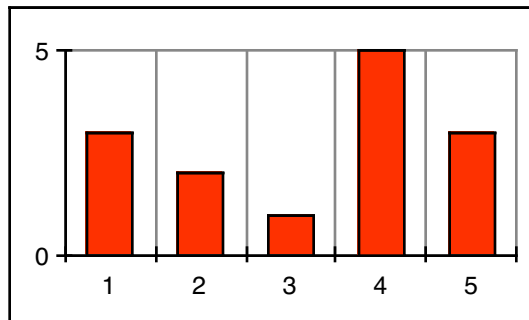
A total of six layout functions are available. All functions appear in pairs in the form of *Open*- and *Close*-functions. They make it possible to:

- define the size of the drawing.
- position charts precisely.
- overlay charts.
- structure complex drawings and charts.

### **OpenDrawing(width;height;type;antialiasing)**

The most important layout function is `OpenDrawing()`. This function defines the width and height of a drawing in pixels and should always be listed as the first function call. For example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 2 1 5 3)
  BarChart()
CloseDrawing()
```



As the 3rd argument, it is possible to define whether a drawing should be created in vector format (*type=0*) or in bitmap format (*type=1*). If there is no 3rd argument, the drawing will be created in vector format as the default. That means in PDF format for Mac OS X and in EMF format for Windows. In addition, in Windows a drawing is created in PDF vector format using *type=-1*. The vector format guarantees high print quality. The optional 4th argument *antialiasing* can be used to control the antialiasing of texts and geometric objects. The following four options are available for this:

- 0: no antialiasing
- 1: antialiasing geometric objects
- 2: antialiasing texts
- 3: antialiasing geometric objects and texts (default)

*Comments:*

*In order to display a drawing, free from distortion and in the proper size in a FileMaker Pro Container field, switch into Layout Mode, set the Graphic Format of the Container field under menu item Format>Graphic... to "Reduce or Enlarge" and switch off the check box "Maintain original proportions".*

*Mac OS X: Due to an error in FileMaker Pro 7 the antialiasing of texts and geometric objects is reliable only in bitmap format (type=1). This error was corrected starting with FileMaker Pro 8.*

Example for vector format:

```
OpenDrawing(200;100;0) // vector format (default)
  ChartData(3 2 1 5 3)
  BarChart()
CloseDrawing()
```

Example for bitmap format:

```
OpenDrawing(200;100;1) // bitmap format
  ChartData(3 2 1 5 3)
  BarChart()
CloseDrawing()
```

The difference between vector and bitmap format can be seen by enlarging the drawing. This can be done simply by using the Zoom-Button on the bottom left of the FileMaker Pro window.



Examples:

```
OpenDrawing(200;100;0;2) // antialiasing fonts
  ChartData(3 2 1 5 3)
  BarChart()
CloseDrawing()
```

```
OpenDrawing(200;100;1;3) // bitmap + antialiasing
  ChartData(3 2 1 5 3)
  BarChart()
CloseDrawing()
```

**CloseDrawing()**

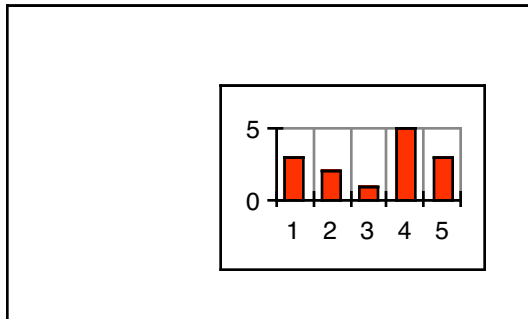
CloseDrawing() is optional, does not have any arguments and can be listed at the end as the last function call.

Output functions like SaveAsJPGFile() can also be placed outside the functions OpenDrawing() and CloseDrawing().

**OpenView(left;top;width;height)**

Views make it possible to divide a drawing into rectangular regions. For example, a series of graphic primitives, such as lines, areas or texts, can be combined within a view; by simply changing the position of the view all graphic components within a view are automatically moved with it. Otherwise the coordinates of each single graphic primitive would have to be changed. That means all coordinates within a view always refer to the upper left-hand corner of the view. For example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  OpenView(80;40;100;70)
    AddFrame(0;0;100;70)
    ChartData(3 2 1 5 3)
    BarChart()
  CloseView()
CloseDrawing()
```



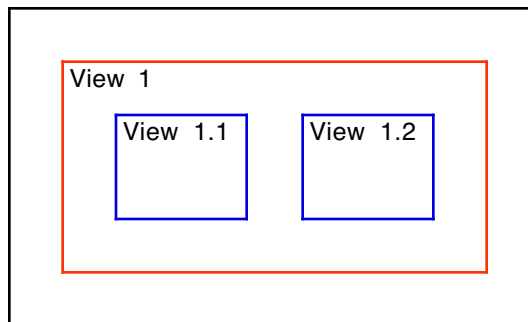
Charts and graphic primitives, which are located partially or completely outside of a view, are clipped. Nested views, i.e. views within a view, are also possible.

Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  OpenView(20;20;160;80)
    AddFrame(0;0;160;80;1;red)
    AddText(3;10;"View 1")
    OpenView(20;20;50;40)
      AddFrame(0;0;50;40;1;blue)
      AddText(3;10;"View 1.1")
    CloseView()
    OpenView(90;20;50;40)
      AddFrame(0;0;50;40;1;blue)
      AddText(3;10;"View 1.2")
    CloseView()
  CloseView()
CloseDrawing()

```



### **CloseView()**

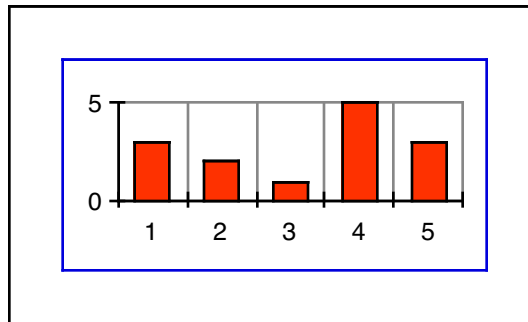
`CloseView()` does not have any arguments and closes the sequence of commands within a view. If only one single view is defined, it is not necessary to use `CloseView()`.

### **OpenChart(left;top;width;height;isPlotArea)**

`OpenChart()` makes it possible to position a chart precisely within a drawing or a view. By using the 5th argument *isPlotArea*, the arguments *left*, *top*, *width* and *height* either define the entire area incl. axes, title and legend (*isPlotArea=off*) or only the actual area of the graph (*isPlotArea=on*).

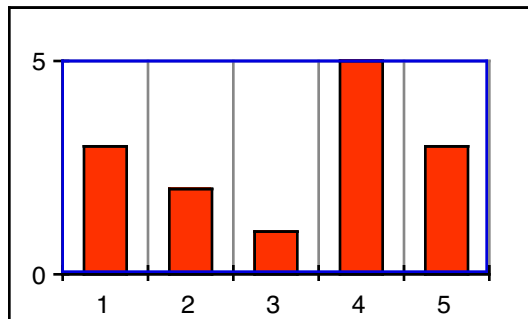
Example for *isPlotArea=off*: (default)

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  OpenChart(20;20;160;80;off)
    ChartData(3 2 1 5 3)
    BarChart()
    AddFrame(20;20;160;80;1;blue)
  CloseChart()
CloseDrawing()
```



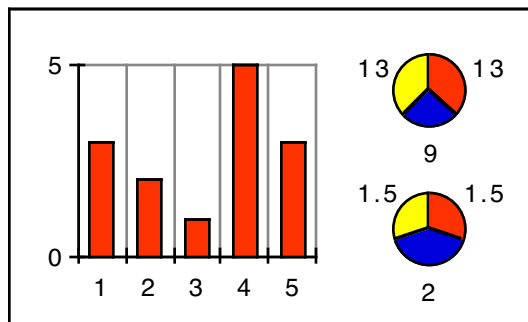
Example for *isPlotArea=on*:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  OpenChart(20;20;160;80;on)
    ChartData(3 2 1 5 3)
    BarChart()
    AddFrame(20;20;160;80;1;blue)
  CloseChart()
CloseDrawing()
```



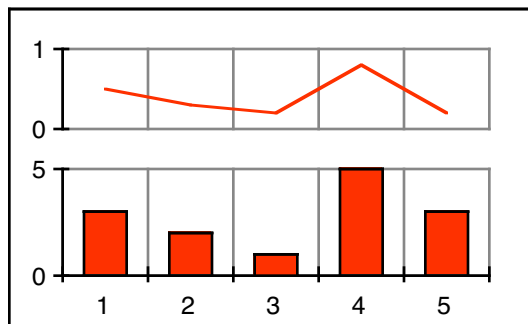
The argument *isPlotArea=off* can be used to position several charts within a drawing. For example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  OpenChart(5;5;120;115;off)
    ChartData(3 2 1 5 3)
    BarChart()
  CloseChart()
  OpenChart(120;0;75;75;off)
    ChartData(13 9 13)
    PieChart(label)
  CloseChart()
  OpenChart(120;50;75;75;off)
    ChartData(1.5 2 1.5)
    PieChart(label)
  CloseChart()
CloseDrawing()
```



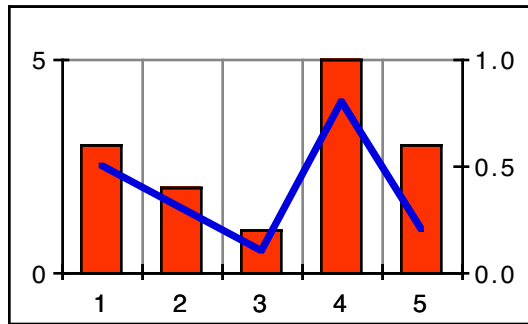
The argument *isPlotArea=on* can be used to precisely align or overlay two or more charts. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  OpenChart(20;15;160;30;on)
    ChartData(0.5 0.3 0.2 0.8 0.2)
    LineChart(;on)
    AxisOptions(x;none) // hide x-axis
  CloseChart()
  OpenChart(20;60;160;40;on)
    ChartData(3 2 1 5 3)
    BarChart()
  CloseChart()
CloseDrawing()
```



```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  OpenChart(20;20;150;80;on)
    ChartData(3 2 1 5 3)
    BarChart()
  CloseChart()
  OpenChart(20;20;150;80;on)
    ChartData(0.5 0.3 0.1 0.8 0.2)
    LineChart(;on)
    LineStyle(1;poly;2;blue)
    AxisOptions(y;;on) // move y-axis to the right
    GridLocation(xy;none) // hide grid
  CloseChart()
CloseDrawing()
```



**CloseChart()**

CloseChart() does not have any arguments and closes the sequence of commands within a chart. If only one single chart is defined, it is not necessary to use CloseChart().

## Data

Six functions are available to enter and manage chart data. They serve to:

- enter chart data
- import and export chart data
- deal with invalid or missing values

### **ChartData(series1;series2...)**

The `ChartData()` function makes it possible to transfer the chart values to `xmCHART`. The numbers are normally copied to the `ChartData()` function from the database fields by means of FileMaker Pro scripts. This is explained in detail in *xmExamples*. Instead of numerical data, dates and times can also be passed to `xmCHART`. This will be explained later in the chapter *Entry of Date and Time*.

Each data series consists of a series of numbers which are separated by spaces, tabs or line feeds. Decimal numbers can be represented by a decimal point or decimal comma. Example:

```
ChartData(-10 2.31 0 -,69 0.01 -8,00 .0)
```

Numbers can also be entered in scientific notation (E-format). Example:

```
ChartData(-1.2e04 0.2E04 .2e-3)
```

Missing values can be indicated by a NULL. Example:

```
ChartData(12 98.3 null 8 Null NULL 7.23 -0.67)
```

Thousands separators are not permitted. Example:

```
ChartData(1,234.56 12.345,67) // invalid!
```

Multiple data series are separated by a semicolon ";". The meaning of the individual data series in `ChartData()` is dependent on the chart to be drawn. For the 1-dimensional chart functions, `AreaChart()`, `BarChart()`, `BoxPlot()`, `GanttChart()`, `Histogram()`, `LineChart()`, `RadarChart()` and `ScatterChart()`, a data series in `ChartData()` corresponds exactly to a chart series. For the 2-dimensional chart functions, `LineChart2D()`, `ScatterChart2D()` and `PolarChart()`, the first data series in `ChartData()` contains the x values, the second data series the y values of the 1st chart series, the 3rd and 4th data series the x and y values of the 2nd chart series, etc.

Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(23 19 10 15 13 6 2; // 1st series
            18 12 19 24 15 15 6) // 2nd series
  LineChart(symbol)
  SymbolStyle(all;bullet;4)
CloseDrawing()
```

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(23 19 10 15 13 6 2; // x-values
            18 12 19 24 15 15 6) // y-values
  LineChart2D(symbol)
  SymbolStyle(all;bullet;4)
CloseDrawing()
```

Special assignments between data series and chart series, such as for high-low or bubble charts, can be found in the *Charts* section.

### **ChartDataOptions(scanDirection)**

As the default, the values of a chart series are entered successively (*scanDirection=xyxy*). Example:

```
ChartData(2 3 -10 5; // 1st series
          3 9 12 -3) // 2nd series
```

However, from time to time, it can be advantageous to enter the data in transposed form, i.e. the rows and columns are switched (*scanDirection=xyxy*). Example:

```
ChartData(2 3;
          3 9;
          -10 12;
          5 -3)
```

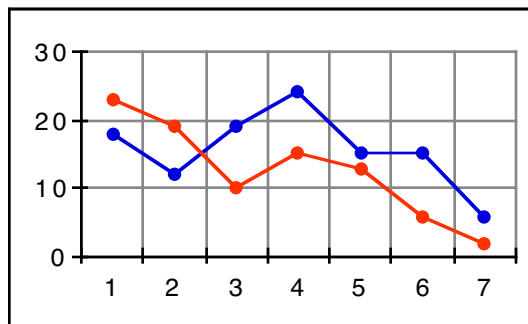
For example, several data series from a sequence of FileMaker Pro records can be transferred in transposed form to `ChartData()` via one single loop. Otherwise, if not in transposed form, a separate loop for each data series is necessary. Please note that the `ChartDataOptions()` function should be entered before the `ChartData()` function.

## Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartDataOptions(xyxy) // before ChartData()!
  ChartData(23 18;
            19 12;
            10 19;
            15 24;
            13 15;
            6 15;
            2 6)
  LineChart(symbol;on)
  SymbolStyle(1;bullet;4)
  SymbolStyle(2;bullet;4)
CloseDrawing()

```



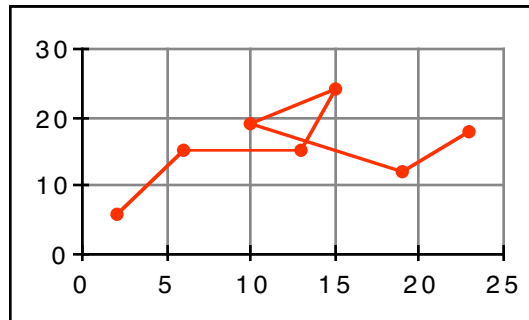
The following two scripts produce the same result:

```

// not transposed (default)
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(23 19 10 15 13 6 2; // x-values
            18 12 19 24 15 15 6) // y-values
  LineChart2D(symbol)
  SymbolStyle(1;bullet;4)
CloseDrawing()

```

```
// transposed
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartDataOptions(xyxy) // before ChartData()!
  ChartData(23 18;
            19 12;
            10 19;
            15 24;
            13 15;
            6 15;
            2 6)
  LineChart2D(symbol)
  SymbolStyle(1;bullet;4)
CloseDrawing()
```



**ChartDataUpperLimits(maxValue1;maxValue2...)**  
**ChartDataLowerLimits(minValue1;minValue2...)**

Both functions, `ChartDataUpperLimits()` and `ChartDataLowerLimits()`, make it possible to define an upper and lower limit for each data series, i.e. all values which are less than *minValue* or greater than *maxValue* are not shown. The functions can be used to draw incomplete data series: missing data, such as missing readings, is substituted by "invalid" values, i.e. by values which are less than *minValue* or greater than *maxValue*. Please note that the `ChartDataUpperLimits()` and `ChartDataLowerLimits()` functions should be entered after the `ChartData()` function.

A different and easier method for passing incomplete series of values is to replace the missing values with NULL. Example:

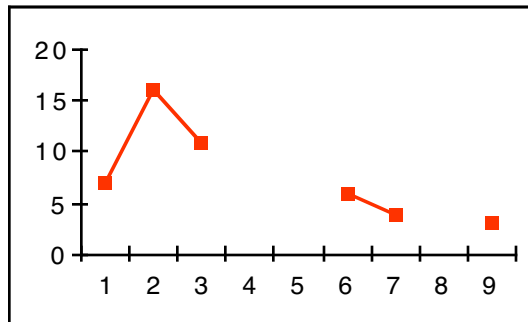
```
ChartData(12 98.3 null 8 Null NULL 7.23 -0.67)
```

## Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  // -1..."missing values"
  ChartData(7 16 11 -1 -1 6 4 -1 3)
  ChartDataLowerLimits(0.0) // after ChartData()!
  LineChart(symbol;on)
  SymbolStyle(1;square;4;;red)
  GridLocation(;none)
CloseDrawing()

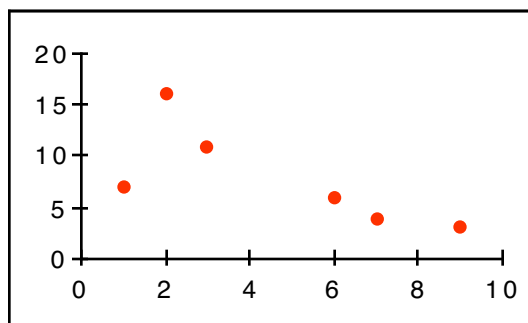
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  // 9999..."missing values"
  ChartData(1 2 3 4 5 6 7 8 9;
            7 16 11 9999 9999 6 4 9999 3)
  ChartDataUpperLimits(1000) // after ChartData()!
  ScatterChart2D()
  SymbolStyle(1;bullet;4)
  GridLocation(;none)
CloseDrawing()

```



**ChartDataRead( fileName;doTranspose;seriesSeparator;  
elementSeparator)**

Instead of taking the chart values directly from FileMaker Pro via ChartData( ) they can also be imported from a text file. In doing so, it is also possible to import very large amounts of data, as the 64kB restriction of FileMaker Pro fields can be evaded.

By using the 1st argument *fileName* the name of the file including an optional file path is defined; details can be found in the *Output* section. As the default, the data is read row by row, i.e. a row corresponds to a data series. If the 2th argument *doTranspose=on* is used, the data is read in columns, i.e. a column corresponds to a data series. By using the arguments *seriesSeparator* and *elementSeparator* user-defined separators can be specified between the series and between the individual values. Unless otherwise defined, series are separated by a line feed "\n" and individual values by a tab character "\t". Decimal numbers can be entered using a decimal point or decimal comma; however, thousands separators are not permitted.

The ChartDataOptions( ) function is ignored in connection with ChartDataRead( ). Examples:

```
ChartDataRead("Macintosh HD/Data/Plotdata.dat")  
ChartDataRead("C:/Programs/Data/plotdata.txt";";";" ")
```

The ChartDataRead( ) function can be positioned as you like, but always within the OpenDrawing( ) and CloseDrawing( ) functions. ChartDataRead( ) can also be called repeatedly. Examples:

```
OpenDrawing(200;120)  
  AddFrame(0;0;200;120)  
  LineChart(symbol;on)  
  SymbolStyle(1;bullet;4)  
  SymbolStyle(2;bullet;4)  
  ChartDataRead("Data")  
CloseDrawing( )
```

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  OpenChart(5;5;120;115;off)
    ChartDataRead("Data_1.txt")
    BarChart()
  CloseChart()
  OpenChart(120;0;75;75;off)
    ChartDataRead("Data_2.txt")
    PieChart(label)
  CloseChart()
  OpenChart(120;50;75;75;off)
    ChartDataRead("Data_3.txt")
    PieChart(label)
  CloseChart()
CloseDrawing()

```

**ChartDataWrite(fileName;fileFlag;creatorType;  
doTranspose;seriesSeparator;elementSeparator;  
formatSpecifier1;formatSpecifier2...)**

By using the `ChartDataWrite()` function it is possible to save the chart values in a text file. The first three arguments, *fileName*, *fileFlag* and *creatorType*, are explained in detail in the *Output* section. As the default, the data is written row by row, i.e. a row corresponds to a data series. If the 4th argument *doTranspose=on* is used, the data is written in columns, i.e. a column corresponds to a data series. Furthermore, user-defined separators can be specified by using the arguments *seriesSeparator* and *elementSeparator*. Unless otherwise defined, series are separated by a line feed "\n" and the individual values by a tab character "\t". As an option, individual format specifiers can be assigned to data series. The format specifiers are repeated periodically if there are fewer format specifiers than data series. If no format specifier is defined, the default format specifier "|u|" is used. All format specifiers incl. numerous examples can be found in *xmReference*. All numbers are exported without a thousands separator; furthermore, all decimal numbers always with a decimal point "." and not with a decimal comma ",". Examples:

```

ChartDataWrite("Macintosh HD/Data/Plots/ExportData";replace)
ChartDataWrite("Data.txt";"ttxt";";";" ";"|i0|";"|f2|")
ChartDataWrite("C:/Programs/Data/export.txt";;on)

```



The `ChartDataWrite()` function can be entered either before or after the `ChartData()` function, but always within the `OpenDrawing()` and `CloseDrawing()` functions. `ChartDataWrite()` can also be called repeatedly. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartDataWrite("Data";replace;;on;;; "|f1|"; "|f2|")
  ChartData(23 19 10 15 13 6 2; // x-values
            18 12 19 24 15 15 6) // y-values
  LineChart2D(symbol)
  SymbolStyle(1;bullet;4)
CloseDrawing()
```

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  OpenChart(5;5;120;115;off)
    ChartData(3 2 1 5 3)
    ChartDataWrite("Data_1.txt";replace)
    BarChart()
  CloseChart()
  OpenChart(120;0;75;75;off)
    ChartData(13 9 13)
    ChartDataWrite("Data_2.txt";replace)
    PieChart(label)
  CloseChart()
  OpenChart(120;50;75;75;off)
    ChartData(1.5 2 1.5)
    ChartDataWrite("Data_3.txt";replace)
    PieChart(label)
  CloseChart()
CloseDrawing()
```

## Entry of Date and Time

Instead of numerical data, dates and times can also be passed to `ChartData()`. The entry of both straight dates, e.g.

```
ChartData(1/1/2009 3/17/2009 3/18/2009 10/22/2009)
```

and straight times are supported. For Example:

```
ChartData(1:00 8:32 14:01:38 23:22:11)
```

However, combined dates and times (so-called time stamps) are also permitted. The date and time are either joined together by an ampersand "&" or put in quotes. Example:

```
ChartData(2009-01-01&11:22 "2009-03-17 18:33:00")
```

Periods, slashes and hyphens can be used to separate the date components entered. Example:

```
ChartData(1.1.2009 4/13/2009 2009-05-22)
```

Colons and apostrophes can be used to enter the times. Typographical apostrophes ( ' ' ) are not allowed. Example:

```
ChartData(0:22 11:22:33 3'13 14'05'22)
```

The 1st argument *dateOrder* of the function `DateTimeOptions()` can be used to predefine the order of day, month and year when entering the date. The following three orders are permitted:

<i>Date order</i>	<i>Constant</i>	<i>Value</i>	<i>Example</i>
year, month, day	ymd	1	2009-08-04
month, day, year	mdy	2 (default)	4/8/2009
day, month, year	dmy	3	4.8.2009

To rule out ambiguities when entering the date, e.g. 4/5/12 or 4/8/2012 (European: August 4, 2012 or American: April 8, 2012), the following rules should be followed:

- Years should always include the century, e.g. 1998 and not 98. If the century is left out, then the two-digit year will automatically be turned into a four-digit year by xmCHART. The conversion can be carried out when the four-digit year, starting with the current year, lies either within the previous 70 years or within the next 30 years. For example, if the current year is 2012, then the years 00, 01, 02 to 42 will be turned into the years 2000, 2001, 2002 to 2042 and the years 43 to 99 into 1943 to 1999. If the current year is 2013, then the years 00, 01, 02 to 43 will be turned into the years 2000, 2001, 2002 to 2043 and the years 44 to 99 into 1944 to 1999.

- If the date is entered in the order of day-month-year or month-day-year, then it is always advisable to specifically predefine the date order by using the function `DateTimeOptions()`. In doing so, it is important to always enter the `DateTimeOptions()` function before the `ChartData()` function. When using the order of year-month-day, it is not absolutely necessary as this order is clear — provided the year is entered with the century.

Examples:

```
ChartData(2009-02-01 2009-04-12) // Feb. 1, Apr. 12
```

```
DateTimeOptions(dmy) // European  
ChartData(1/2/2009 12/4/2009) // Feb. 1, Apr. 12
```

```
DateTimeOptions(mdy) // American  
ChartData(1/2/2009 12/4/2009) // Jan. 2, Dec. 4
```

## Charts

xmCHART makes a variety of different charts available which can be divided into the following categories:

- scatter charts (1 and 2-dimensional)
- line charts (1 and 2-dimensional)
- area charts (1 and 2-dimensional)
- bar charts (2 and 3-dimensional)
- bubble charts (1 and 2-dimensional)
- pie charts (2 and 3-dimensional)
- polar and radar charts
- financial business charts
- statistical charts

The charts can be created and varied by using numerous options. In the following all types of charts are explained in detail together with several examples.

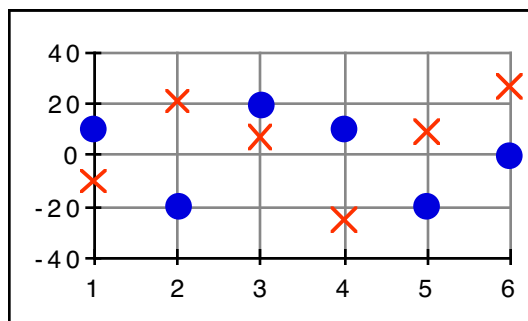
### Scatter charts:

Two functions are available for creating scatter charts:

#### **ScatterChart (appearanceConstants;doShiftIntervals)**

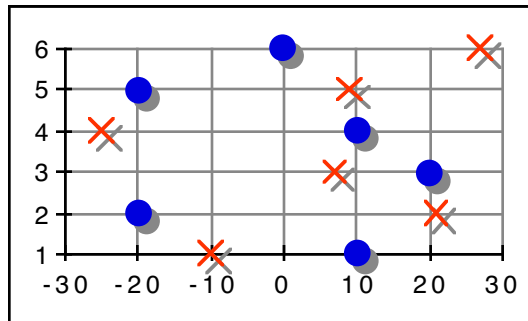
The ScatterChart() function serves to draw one-dimensional scatter charts. Example:

```
OpenDrawing(200;120)
AddFrame(0;0;200;120)
ChartData(-10 21 7 -25 9 27; // 1st series
          10 -20 20 10 -20 0) // 2nd series
ScatterChart()
CloseDrawing()
```



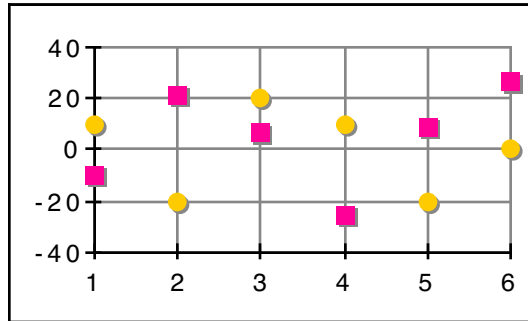
The 1st argument *appearanceConstants* makes it possible to rotate the chart 90 degrees (*appearanceConstants=horizontal*), to add shadow (*appearanceConstants=shadow*) and to add labels (*appearanceConstants=label*) to the symbols. All options can be combined arbitrarily by using a plus sign "+". Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(-10 21 7 -25 9 27; // 1st series
            10 -20 20 10 -20 0) // 2nd series
  ScatterChart(shadow+horizontal)
CloseDrawing()
```



The symbols and shadows can be varied by using the style functions *SymbolStyle()* and *ShadowStyle()* and the labels by using the four style functions *LabelTexts()*, *LabelStyle()*, *LabelBackground()* and *LabelOptions()*. All style functions are discussed in detail in the *Styles* section. Examples:

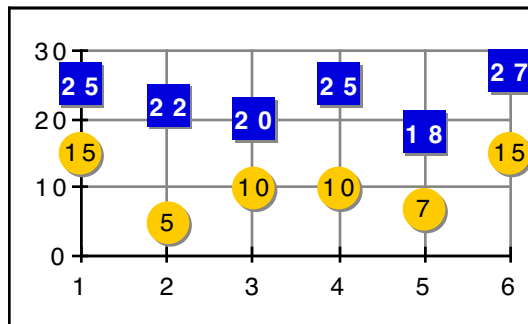
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(-10 21 7 -25 9 27; // 1st series
            10 -20 20 10 -20 0) // 2nd series
  ScatterChart(shadow)
  SymbolStyle(1;square;6;;purple)
  SymbolStyle(2;bullet;6;;darkYellow)
  ShadowStyle(all;1;gray)
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(25 22 20 25 18 27; // 1st series
            15 5 10 10 7 15) // 2nd series
  ScatterChart(shadow+label)
  SymbolStyle(1;square;15;;blue)
  SymbolStyle(2;bulet;15;;darkYellow)
  ShadowStyle(all;1;gray)
  LabelOptions(all;centerCenter)
  LabelStyle(1;;;bold;white)
CloseDrawing()

```

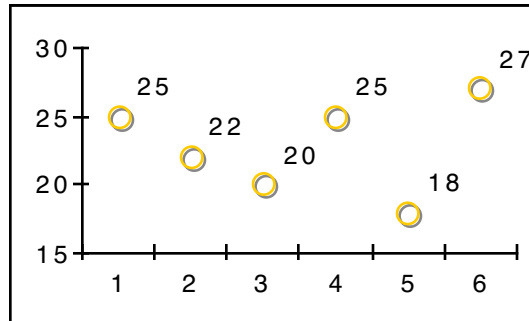


For one-dimensional charts it is sometimes advantageous to move the symbols half an interval width so that they do not lie on the interval borders but rather in the middle of the intervals. This can be done by activating the 2nd argument *doShiftIntervals=on*. Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(25 22 20 25 18 27)
  ScatterChart(shadow+label;on)
  SymbolStyle(1;circle;8;;darkYellow)
  ShadowStyle(all;1;gray)
  GridLocation(all;none)
CloseDrawing()

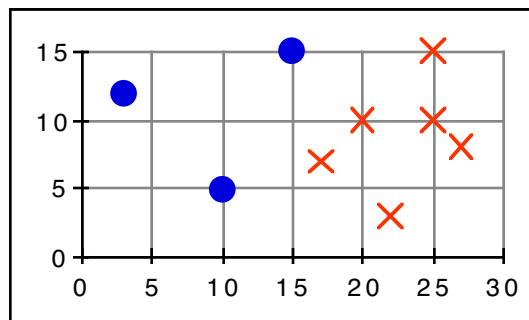
```



### ScatterChart2D(*appearanceConstants*)

The `ScatterChart2D()` function makes it possible to draw 2-dimensional scatter charts. The 1st data series in the `ChartData()` function provides the x-values, the 2nd data series the y-values for the first symbol series, the 3rd and 4th data series in `ChartData()` the x and y-values for the second symbol series, etc. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(25 22 20 25 17 27; // 1st series (x-values)
            15 3 10 10 7 8; // 1st series (y-values)
            3 10 15; // 2nd series (x-values)
            12 5 15) // 2nd series (y-values)
  ScatterChart2D()
CloseDrawing()
```



As for one-dimensional scatter charts, shadow (*appearanceConstants=shadow*) and labels (*appearanceConstants=label*) can be added to the symbols by using the argument *appearanceConstants*. The options can be combined by using a plus sign "+".

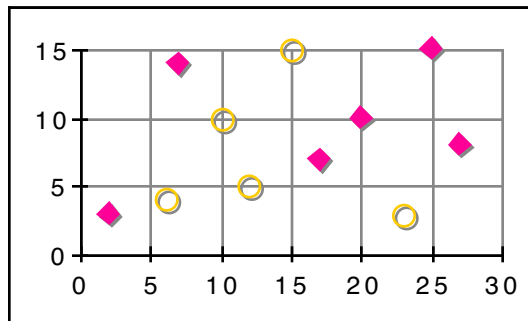
Rotating (*appearanceConstants=horizontal*) is not supported for two-dimensional charts as this is simply possible by switching the data series in `ChartData()`.

Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(25 2 20 7 17 27; // 1st series (x-values)
            15 3 10 14 7 8; // 1st series (y-values)
            23 10 15 12 6; // 2nd series (x-values)
            3 10 15 5 4) // 2nd series (y-values)
  ScatterChart2D(shadow)
  SymbolStyle(1;diamond;7;;purple)
  SymbolStyle(2;circle;8;;darkYellow)
  ShadowStyle(all;1;gray)
CloseDrawing()

```



### Line charts:

Two functions are available for creating line charts:

#### **LineChart(appearanceConstants;doShiftIntervals)**

The `LineChart()` function serves to draw one-dimensional line charts.

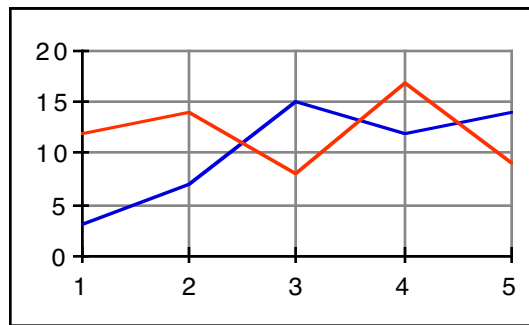
Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(12 14 8 17 9; // 1st series
            3 7 15 12 14) // 2nd series
  LineChart()
CloseDrawing()

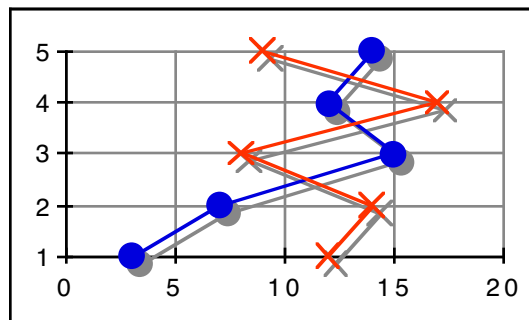
```





The 1st argument *appearanceConstants* makes it possible to rotate the chart 90 degrees (*appearanceConstants=horizontal*) and to add symbols (*appearanceConstants=symbol*), shadow (*appearanceConstants=shadow*) and labels (*appearanceConstants=label*) to the lines. All options can be combined by using a plus sign "+". Example:

```
OpenDrawing(200;120)
AddFrame(0;0;200;120)
ChartData(12 14 8 17 9; // 1st series
          3 7 15 12 14) // 2nd series
LineChart(shadow+horizontal+symbol)
CloseDrawing()
```



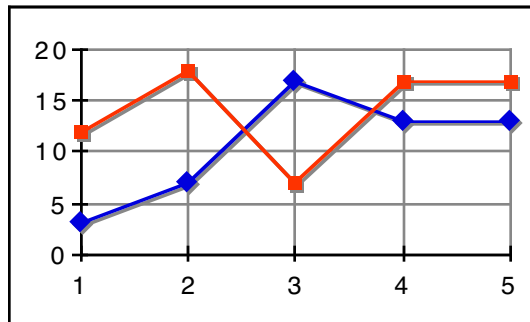
The lines, symbols and shadows can be varied by using the style functions *LineStyle()*, *SymbolStyle()* and *ShadowStyle()* and the labels by using the four style functions *LabelTexts()*, *LabelStyle()*, *Label-Background()* and *LabelOptions()*. All style functions are discussed in detail in the *Styles* section.

Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(12 18 7 17 17; // 1st series
            3 7 17 13 13) // 2nd series
  LineChart(shadow+symbol)
  SymbolStyle(1;square;4;;red)
  SymbolStyle(2;diamond;6;;blue)
  ShadowStyle(all;1;gray)
CloseDrawing()

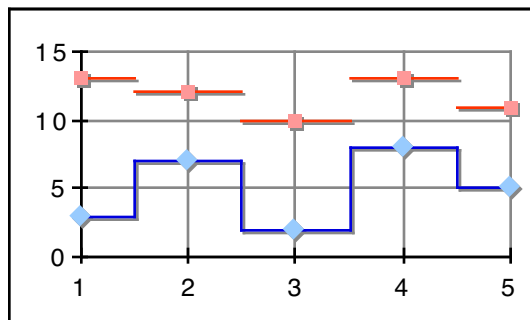
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(13 12 10 13 11; // 1st series
            3 7 2 8 5) // 2nd series
  LineChart(shadow+symbol)
  LineStyle(1;jump)
  LineStyle(2;step)
  SymbolStyle(1;square;4;;lightRed)
  SymbolStyle(2;diamond;6;;lightBlue)
  ShadowStyle(all;1;gray)
CloseDrawing()

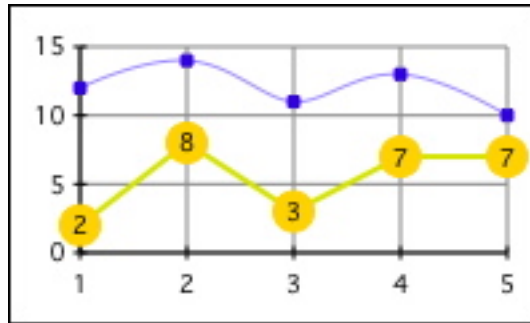
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(12 14 11 13 10; 2 8 3 7 7)
  LineChart(symbol+label)
  LineStyle(1;smooth;1;150 150 255)
  LineStyle(2;poly;2;220 220 0)
  SymbolStyle(1;square;4;;blue)
  SymbolStyle(2;bulet;15;;darkYellow)
  LabelTexts(1;" ") // hide labels
  LabelOptions(2;centerCenter)
CloseDrawing()

```

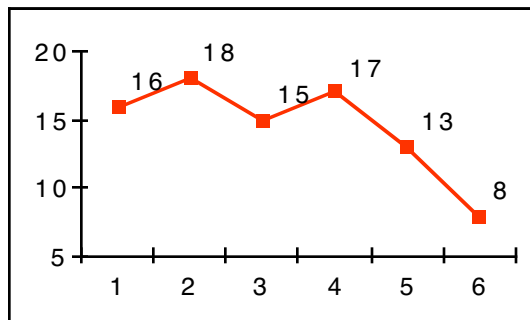


For one-dimensional line charts it is possible to move the polygon points half an interval width so that they do not lie on the interval borders but rather in the middle of the intervals. This can be done by activating the 2nd argument *doShiftIntervals=on*. Example:

```

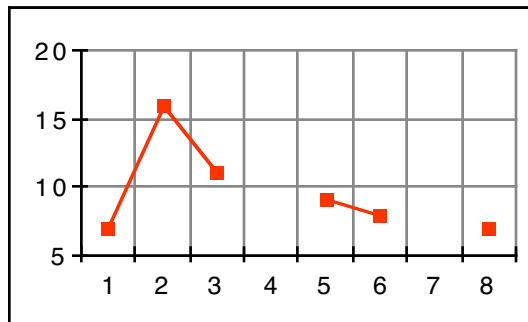
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(16 18 15 17 13 8)
  LineChart(symbol+label;on)
  SymbolStyle(1;square;4;;red)
  GridLocation(all;none)
CloseDrawing()

```



In addition, it is possible to draw discontinuous lines by **replacing** the missing values in `ChartData()` with `NULL`. Example:

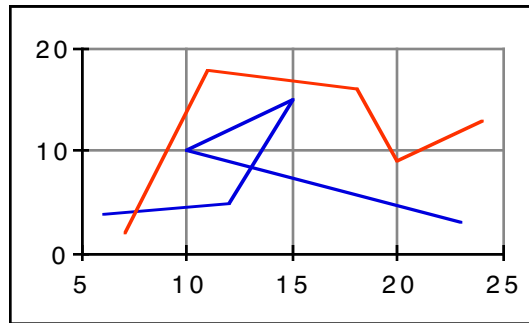
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 16 11 null 9 8 null 7)
  LineChart(symbol;on)
  SymbolStyle(1;square;4;;red)
CloseDrawing()
```



### **LineChart2D(appearanceConstants)**

The `LineChart2D()` function makes it possible to draw two-dimensional line charts. The 1st data series in the `ChartData()` function provides the x-values, the 2nd data series the y-values for the first line series, the 3rd and 4th data series in `ChartData()` the x and y-values for the second line series, etc. Example:

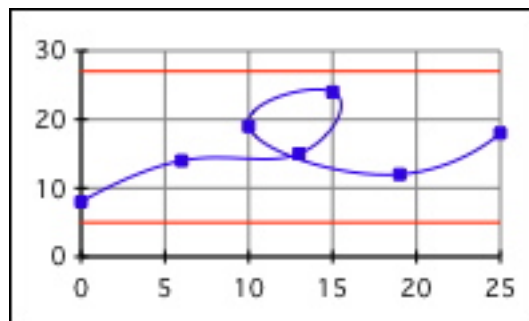
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 7 11 18 20 24; // 1st series (x-values)
            2 18 16 9 13; // 1st series (y-values)
            23 10 15 12 6; // 2nd series (x-values)
            3 10 15 5 4) // 2nd series (y-values)
  LineChart2D()
CloseDrawing()
```



As for one-dimensional line charts, symbols (*appearanceConstants=symbol*), shadow (*appearanceConstants=shadow*) and labels (*appearanceConstants=label*) can also be added to the lines by using the argument *appearanceConstants*. The options can be combined arbitrarily by using a plus sign "+".

Rotating (*appearanceConstants=horizontal*) is not supported for two-dimensional charts as this is simply possible by switching the data series in `ChartData()`. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(25 19 10 15 13 6 0; // x-values
            18 12 19 24 15 14 8; // y-values
            0 25; 27 27; // upper border line
            0 25; 5 5) // lower border line
  LineChart2D(symbol)
  LineStyle(all;poly;1;red)
  LineStyle(1;smooth;1;blue)
  SymbolStyle(all;none)
  SymbolStyle(1;square;4;;blue)
CloseDrawing()
```



**Area charts:**

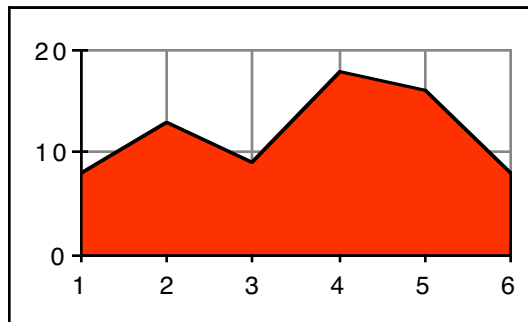
Area charts can be drawn both one and two-dimensionally.

**AreaChart(*appearanceConstants*;*doShiftIntervals*)**

The AreaChart ( ) function serves to draw one-dimensional area charts.

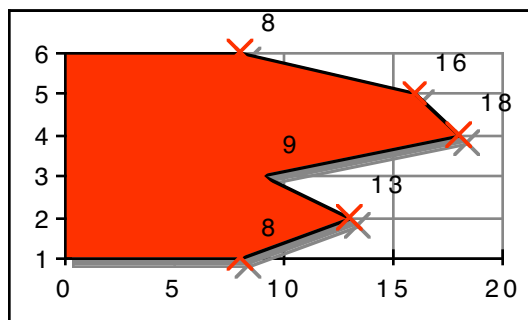
Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(8 13 9 18 16 8)
  AreaChart()
CloseDrawing()
```



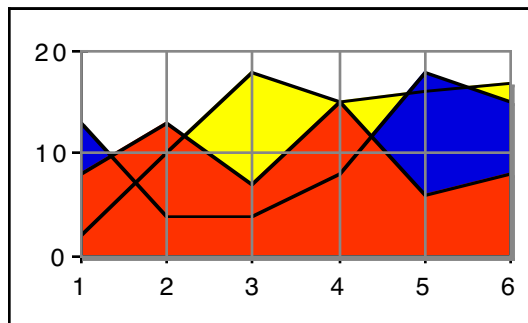
The 1st argument *appearanceConstants* makes it possible to rotate the chart 90 degrees (*appearanceConstants=horizontal*) and to add symbols (*appearanceConstants=symbol*), shadow (*appearanceConstants=shadow*) and labels (*appearanceConstants=label*) to the areas. All options can be combined arbitrarily by using a plus sign "+". Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(8 13 9 18 16 8)
  AreaChart(symbol+label+horizontal+shadow)
CloseDrawing()
```

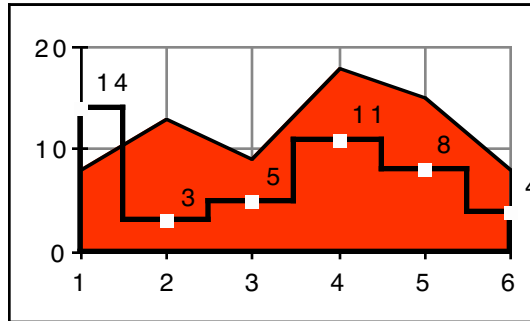


The fills, borders, symbols and shadows can be varied by using the style functions `FillStyle()`, `PictureStyle()`, `BorderStyle()`, `SymbolStyle()` and `ShadowStyle()` and the labels by using the four style functions `LabelTexts()`, `LabelStyle()`, `LabelBackground()` and `LabelOptions()`. All style functions are discussed in detail in the *Styles* section. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 8 13 7 15 6 8;
            13 4 4 8 18 15;
            2 10 18 15 16 17)
  AreaChart(shadow)
  ShadowStyle(all;1;gray)
  GridLocation(all;front)
CloseDrawing()
```



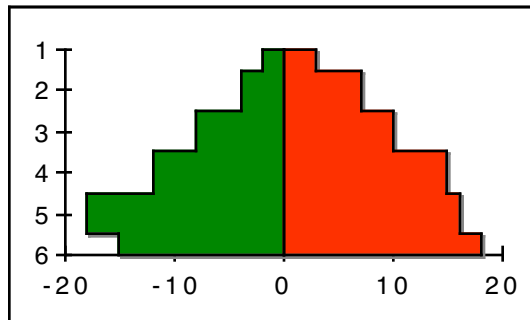
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 8 13 9 18 15 8; // 1st series
            14 3 5 11 8 4) // 2nd series
  AreaChart(symbol+label)
  SymbolStyle(1;none)
  LabelTexts(1;" ") // hide labels
  BorderStyle(2;step;2)
  FillStyle(2;;transparent)
  SymbolStyle(2;square;4;;white)
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 3  7 10 15 16 18; // 1st series
            -2 -4 -8 -12 -18 -15) // 2nd series
  AreaChart(shadow+horizontal)
  BorderStyle(all;step)
  FillStyle(2;green)
  ShadowStyle(all;1;gray)
  GridLocation(all;none)
  ScalingOptions(y;on) // y-scaling top to bottom
CloseDrawing()

```



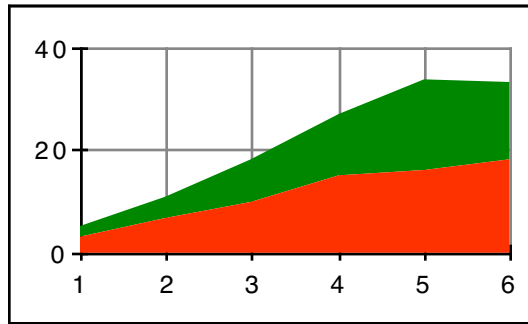
Furthermore, one-dimensional area charts can be stacked (*appearanceConstants=stacked*) and drawn proportionally (*appearanceConstants=proportional*). Proportional charts are automatically stacked. The labeling of stacked charts is explained in detail in the *Styles* section. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3  7 10 15 16 18; // 1st series
            2  4  8 12 18 15) // 2nd series
  AreaChart(stacked)
  BorderStyle(all;none)
  FillStyle(2;green)
CloseDrawing()

```

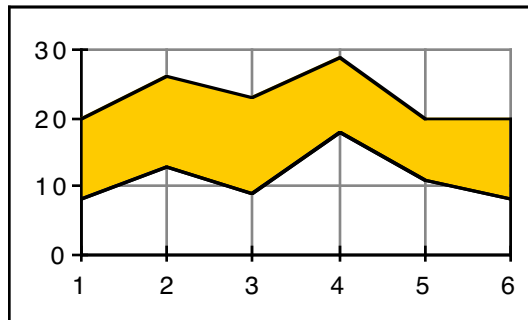




```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 8 13 9 18 11 8; // 1st series
            12 13 14 11 9 12) // 2nd series
  AreaChart(stacked)
  FillStyle(1;;transparent)
  FillStyle(2;darkYellow)
CloseDrawing()

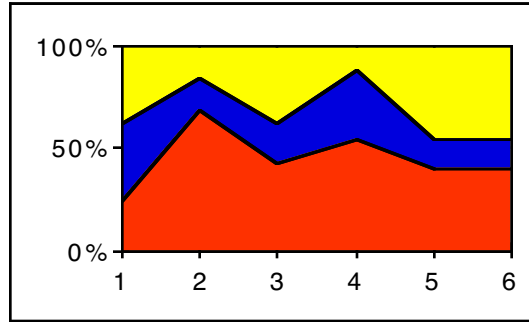
```



```

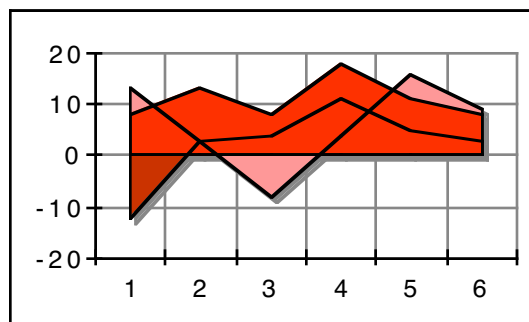
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 8 13 9 18 14 8; // 1st series
            12 3 4 11 5 3; // 2nd series
            12 3 8 4 16 9) // 3rd series
  AreaChart(stacked+proportional)
  Scaling(y;percent)
CloseDrawing()

```



For one-dimensional charts it is possible to move the polygon points half an interval width so that they do not lie on the interval borders but rather in the middle of the intervals. This can be done by activating the 2nd argument *doShiftIntervals=on*. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 8 13 8 18 11 8; // 1st series
            -12 3 4 11 5 3; // 2nd series
            13 3 -8 4 16 9) // 3rd series
  AreaChart(shadow;on)
  FillStyle(2;darkRed)
  FillStyle(3;lightRed)
  ShadowStyle(all;2;gray)
CloseDrawing()
```

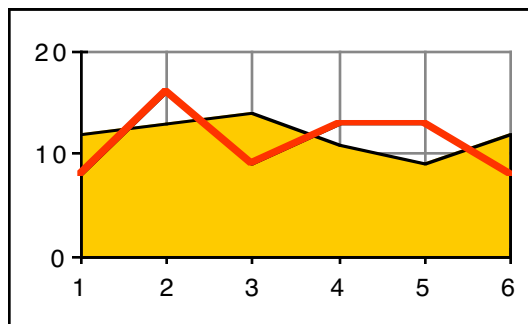


**AreaChartOptions(useLineStyle;referenceValue;  
splitPosNegStacks)**

As an option, a line can be added to the border by activating the argument *useLineStyle=on*. Unlike the default border, which covers the entire area, i.e. including the side borders and the base line, the line will only be drawn between the chart values. This can be used, for example,

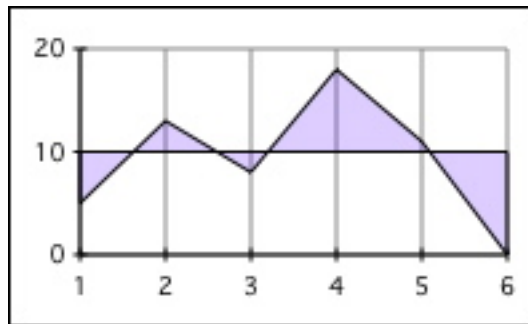
to draw an area and line chart simultaneously; the fill of one series is made invisible. The appearance of the line can be controlled by the `LineStyle()` function. The 2nd argument *shape* in the `LineStyle()` function, which normally serves to control the shape of the curve, is ignored in this context. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 8 16 9 13 13 8; // line
            12 13 14 11 9 12) // area
  AreaChart()
  AreaChartOptions(on) // after AreaChart()!
  LineStyle(1;;2;red)
  LineStyle(2;;0)
  FillStyle(1;;transparent) // no fill
  FillStyle(2;darkYellow)
CloseDrawing()
```



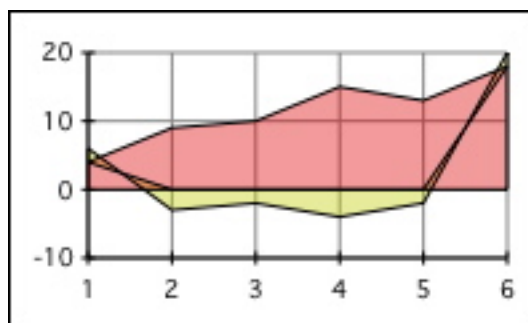
The 2nd argument *referenceValue* makes it possible to define the base line of the areas. As the default, all areas start at zero, i.e. *referenceValue=0*. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 13 8 18 11 0)
  AreaChart()
  AreaChartOptions(;10) // after AreaChart()!
  FillStyle(1;50 0 255 50)
CloseDrawing()
```



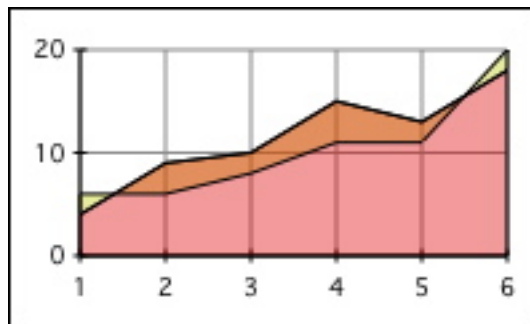
The 3rd argument *splitPosNegStacks* can be used to control the stacking of negative values. As the default, positive and negative values are stacked separately (*splitPosNegStacks=on*), i.e. positive values are added up "above" the reference line, negative values "below" it. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4  9  10 15 13 18;
            2 -3 -2 -4 -2  2)
  AreaChart(stacked)
  AreaChartOptions(;;on) // after AreaChart()!
  FillStyle(1;200  0 0 100)
  FillStyle(2;200 200 0 100)
CloseDrawing()
```



If *splitPosNegStacks=off* is set, positive and negative values will not be stacked separately. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4  9  10 15 13 18;
            2 -3 -2 -4 -2  2)
  AreaChart(stacked)
  AreaChartOptions(;;off) // after AreaChart()!
  FillStyle(1;200  0 0 100)
  FillStyle(2;200 200 0 100)
CloseDrawing()
```

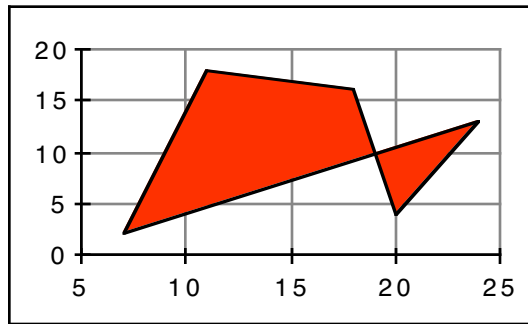


Please note that the `AreaChartOptions()` function should be listed after the `AreaChart()` function. This rule generally applies: *chart options should always be listed after the actual chart function.*

### **AreaChart2D(appearanceConstants;fillAreaToAxis)**

The `AreaChart2D()` function makes it possible to draw two-dimensional area charts. The 1st data series in the `ChartData()` function provides the x-values, the 2nd data series the y-values for the first area, the 3rd and 4th data series in `ChartData()` the x and y-values for the second area, etc. Example:

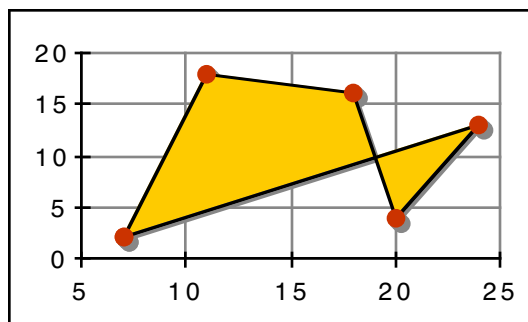
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 7 11 18 20 24 7; // x-values
            2 18 16  4 13 2) // y-values
  AreaChart2D()
CloseDrawing()
```



As for one-dimensional area charts, symbols (*appearanceConstants=symbol*), shadow (*appearanceConstants=shadow*) and labels (*appearanceConstants=label*) can also be added to the areas by using the argument *appearanceConstants*. The options can be combined arbitrarily by using a plus sign "+".

Rotating (*appearanceConstants=horizontal*) is not supported for two-dimensional charts as this is simply possible by switching the data series in `ChartData()`. Examples:

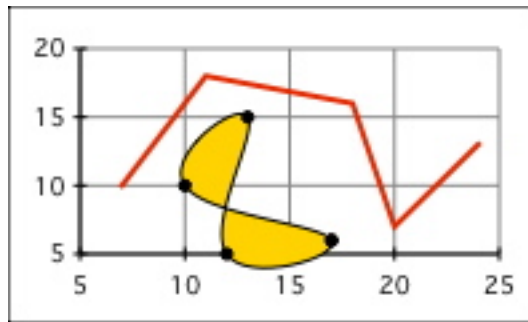
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 7 11 18 20 24 7; // x-values
            2 18 16  4 13 2) // y-values
  AreaChart2D(symbol+shadow)
  FillStyle(1;darkYellow)
  ShadowStyle(1;1)
  SymbolStyle(1;bullet;6;;darkRed)
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 7 11 18 20 24; // 1st series (x-values)
            10 18 16 7 13; // 1st series (y-values)
            17 10 13 12 17; // 2nd series (x-values)
            6 10 15 5 6) // 2nd series (y-values)
  AreaChart2D(symbol)
  FillStyle(1;;transparent)
  FillStyle(2;darkYellow)
  BorderStyle(1;poly;2;darkRed)
  BorderStyle(2;smooth)
  SymbolStyle(1;none)
  SymbolStyle(2;bullet;4;;black)
CloseDrawing()

```



By using the 2nd argument *fillAreaToAxis*, it is possible to extend the fill to the x-axis and/or y-axis. The following three options are available:

*fillAreaToAxis=0*: no fill to the axis (default)

*fillAreaToAxis=1*: fill to the x-axis

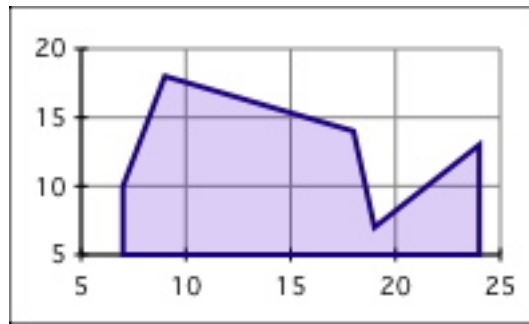
*fillAreaToAxis=2*: fill to the y-axis

Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 7 9 18 19 24; // x-values
            10 18 14 7 13) // y-values
  AreaChart2D(;1)
  FillStyle(1;50 0 200 50)
  BorderStyle(1;poly;2;darkBlue)
CloseDrawing()

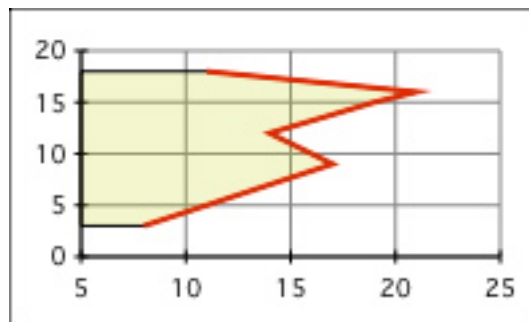
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(8 17 14 19 21 11; // x-values
            3 9 12 15 16 18) // y-values
  AreaChart2D(;2)
  AreaChartOptions(on)
  FillStyle(1;200 200 0 50)
  BorderStyle(1;3;1;)
  LineStyle(1;;2;darkRed)
CloseDrawing()

```





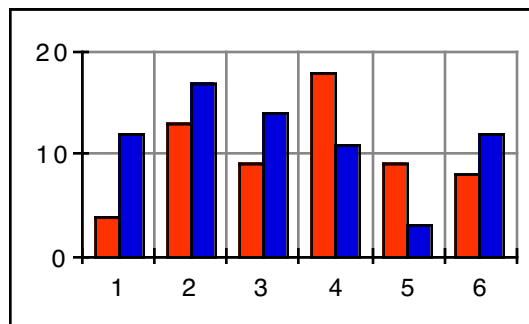
**Bar charts:**

Bar charts can be drawn both one and two-dimensionally.

**BarChart(*appearanceConstants*;*categoryGap*;*seriesGap*;  
barDepth)**

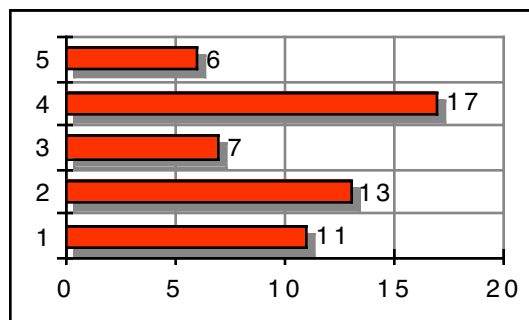
The BarChart() function serves to draw bar charts. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13  9 18 9  8; // "red" series
            12 17 14 11 3 12) // "blue" series
  BarChart()
CloseDrawing()
```



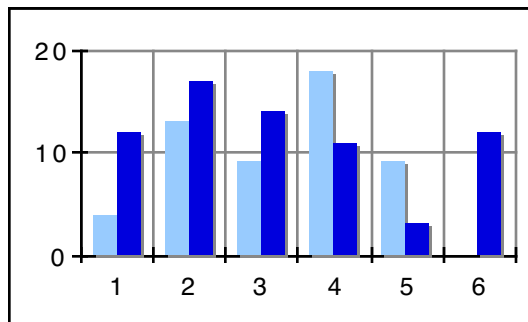
The 1st argument *appearanceConstants* makes it possible to rotate the chart 90 degrees (*appearanceConstants=horizontal*) and to add symbols (*appearanceConstants=symbol*), shadow (*appearanceConstants=shadow*) and labels (*appearanceConstants=label*) to the bars. All options can be combined by using a plus sign "+". Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(11 13 7 17 6)
  BarChart(label+shadow+horizontal)
CloseDrawing()
```

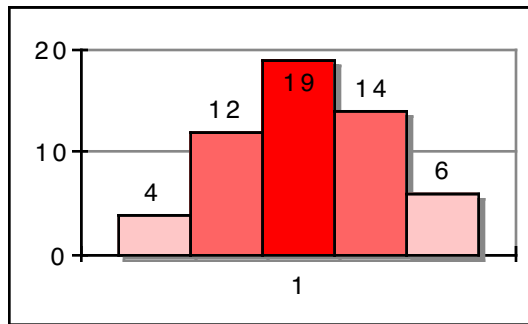


The fills, borders, symbols and shadows can be varied by using the style functions `FillStyle()`, `PictureStyle()`, `BorderStyle()`, `SymbolStyle()` and `ShadowStyle()` and the labels by using the four style functions `LabelTexts()`, `LabelStyle()`, `LabelBackground()` and `LabelOptions()`. All style functions are discussed in detail in the *Styles* section. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13  9 18 9;
            12 17 14 11 3 12)
  BarChart(shadow)
  FillStyle(1;lightBlue)
  BorderStyle(all;none)
  ShadowStyle(all;1;gray)
CloseDrawing()
```



```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4; 12; 19; 14; 6)
  BarChart(shadow+label)
  FillStyle(1;255 200 200)
  FillStyle(2;255 100 100)
  FillStyle(3;255 0 0)
  FillStyle(4;255 100 100)
  FillStyle(5;255 200 200)
  ShadowStyle(all;2;gray)
CloseDrawing()
```

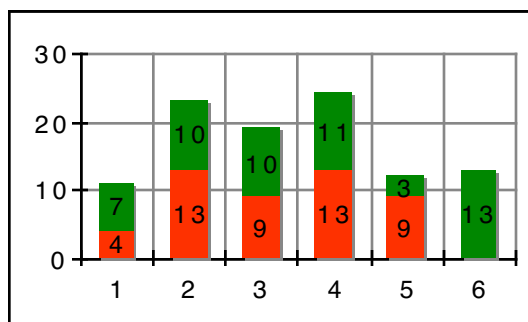


Furthermore, bar charts can be stacked (*appearanceConstants=stacked*) and drawn proportionally (*appearanceConstants=proportional*). Proportional charts are automatically stacked. The labeling of stacked charts is explained in detail in the *Styles* section. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 13 9 13 9;    // 1st series
            7 10 10 11 3 13) // 2nd series
  BarChart(shadow+stacked+label)
  FillStyle(2;green)
  BorderStyle(all;none)
  ShadowStyle(all;1;gray)
CloseDrawing()

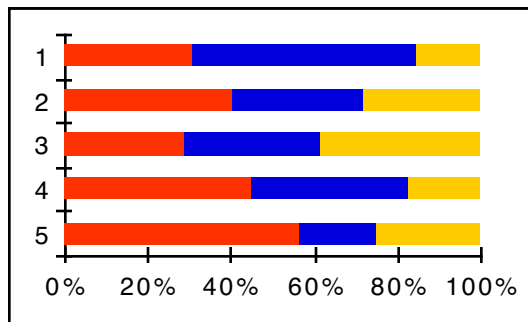
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 13 9 13 9; // 1st series
            7 10 10 11 3; // 2nd series
            2 9 12 5 4) // 3rd series
  BarChart(horizontal+proportional)
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
  GridLocation(all;none)
  Scaling(x;percent)
  ScalingOptions(y;on) // y-scaling top to bottom
CloseDrawing()

```

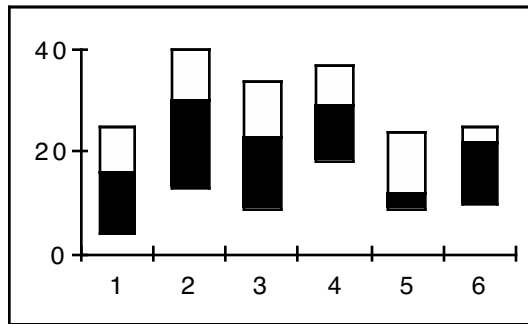


In addition, it is also possible to create floating bar charts by making the first data series invisible. Example:

```

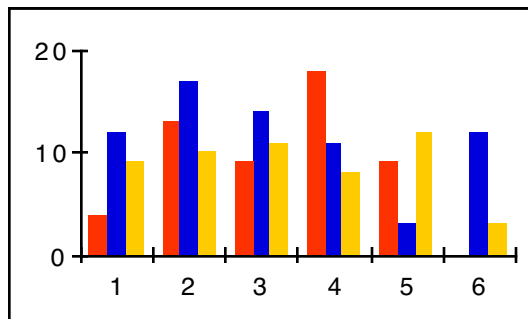
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 9 18 9 10;
            12 17 14 11 3 12;
            9 10 11 8 12 3)
  BarChart(stacked)
  BorderStyle(1;none)
  FillStyle(1;;transparent)
  FillStyle(2;black)
  FillStyle(3;white)
  GridLocation(all;none)
CloseDrawing()

```



By using both arguments *categoryGap* and *seriesGap*, the space between the bars can be controlled. Both arguments are to be entered in percent of the bar width. As the default, the space between two bar categories is exactly one bar width, i.e. *categoryGap*=100. Example:

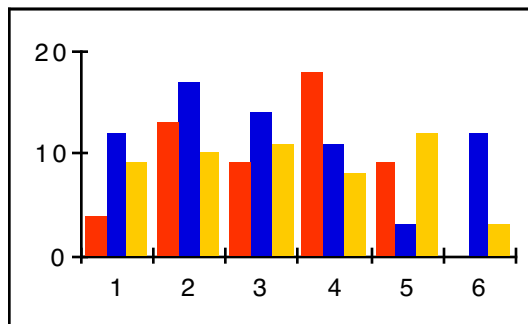
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 9 18 9;
             12 17 14 11 3 12;
             9 10 11 8 12 3)
  BarChart(;100) // default category gap
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
  GridLocation(all;none)
CloseDrawing()
```



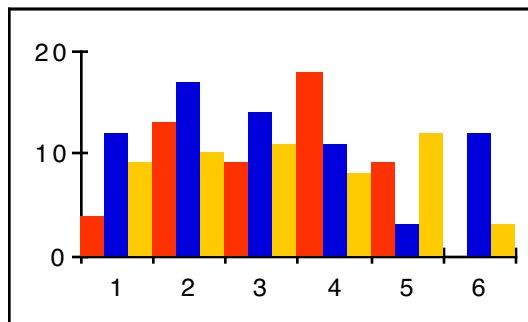
The space between the categories is, for example, halved by using *categoryGap*=50, or completely suppressed by using *categoryGap*=0.

Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 9 18 9;
             12 17 14 11 3 12;
             9 10 11 8 12 3)
  BarChart(;50)
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
  GridLocation(all;none)
CloseDrawing()
```

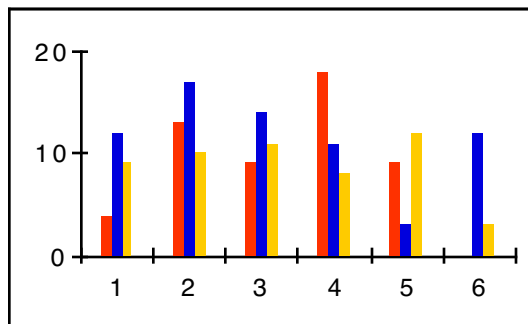


```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 9 18 9;
             12 17 14 11 3 12;
             9 10 11 8 12 3)
  BarChart(;0)
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
  GridLocation(all;none)
CloseDrawing()
```



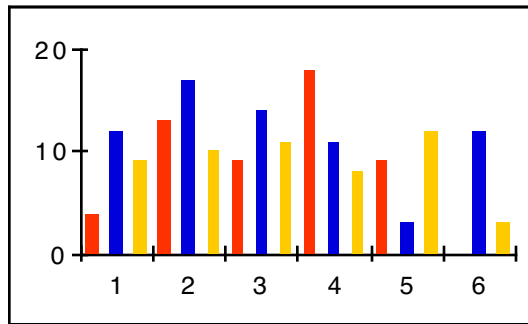
For example, by using *categoryGap=400*, the space between two groups of bars is enlarged by four times the bar width. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 9 18 9;
            12 17 14 11 3 12;
            9 10 11 8 12 3)
  BarChart(;400)
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
  GridLocation(all;none)
CloseDrawing()
```



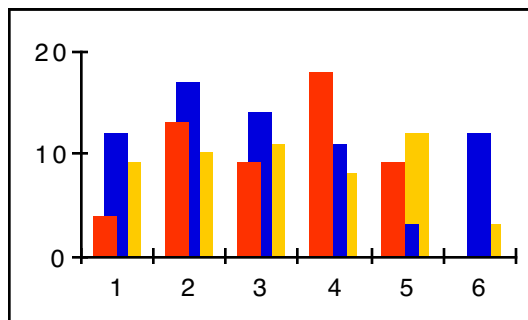
For non-stacked bars there are no spaces between the series as the default (*seriesGap=0*). For example, when *seriesGap=100* a space is inserted between the individual series. The size of the space corresponds to the bar width when *seriesGap=100*. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 9 18 9;
            12 17 14 11 3 12;
            9 10 11 8 12 3)
  BarChart(;;100)
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
  GridLocation(all;none)
CloseDrawing()
```



For non-stacked bars a negative *seriesGap* produces partial or complete overlapping of bars. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 9 18 9;
            12 17 14 11 3 12;
            9 10 11 8 12 3)
  BarChart(;;-50)
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
  GridLocation(all;none)
CloseDrawing()
```

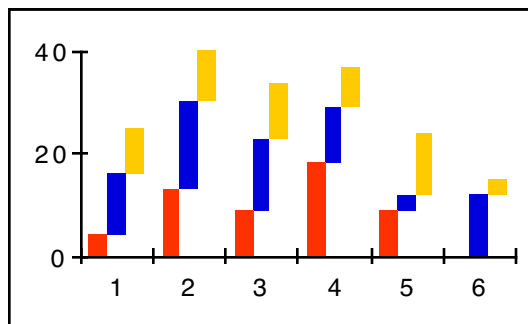


Since stacked bars are not arranged next to each other but rather on top of each other, the default *seriesGap* is -100. When the *seriesGap* is not equal to -100, for example, when *seriesGap*=0, the stacked bars are offset.



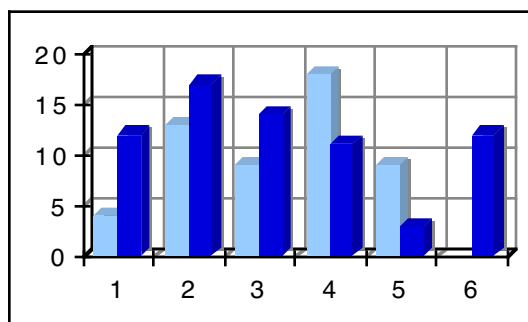
Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 9 18 9;
            12 17 14 11 3 12;
            9 10 11 8 12 3)
  BarChart(stacked;;0)
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
  GridLocation(all;none)
CloseDrawing()
```



The 4th argument *barDepth* can be used to control the three-dimensional drawing of bars. The bar depth is to be entered in percentage of the bar width. Examples:

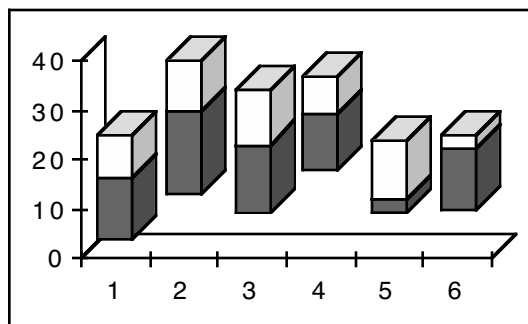
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 9 18 9;
            12 17 14 11 3 12)
  BarChart(shadow;;;50)
  FillStyle(1;lightBlue)
  BorderStyle(all;none)
  ShadowStyle(all;1;gray)
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 9 18 9 10;
            12 17 14 11 3 12;
            9 10 11 8 12 3)
  BarChart(stacked;;;100)
  BorderStyle(1;none)
  FillStyle(1;;transparent)
  FillStyle(2;darkGray)
  FillStyle(3;white)
  GridLocation(all;none)
CloseDrawing()

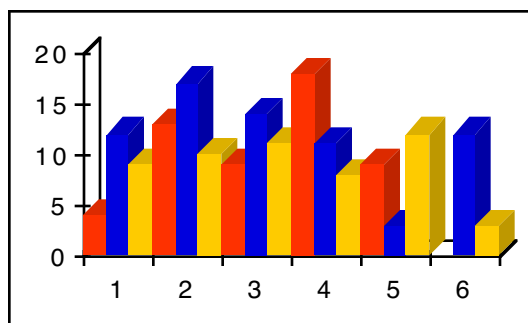
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 9 18 9;
            12 17 14 11 3 12;
            9 10 11 8 12 3)
  BarChart(;0;;100)
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
  GridLocation(all;none)
CloseDrawing()

```



### **BarChart2D(appearanceConstants;categoryGap;seriesGap; barDepth)**

The BarChart2D() function serves to draw bars along the x-axis (time axis). The first data series in ChartData() contains the x values (usually date/time values), the second data series the y values of the 1st chart series, the 3rd and 4th data series the x and y values of the 2nd chart series, etc. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(2009-11-30 2009-12-01 2009-12-04
            2009-12-06 2009-12-07 2009-12-08;
            12 17 14 11 3 12)
  BarChart2D()
  FillStyle(1;200 20 50)
  BorderStyle(all;none)
  AxisMajorTickLabelStyle(x;;;;;-90)
CloseDrawing()
```

The BarChart2D() function is set up exactly the same as the BarChart() function.

### **BarChartOptions(showConnectingLines;referenceValue; makeColorSplit)**

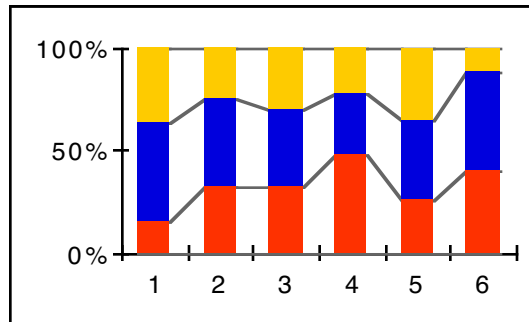
By activating the 1st argument *showConnectingLines=on*, the (in this case normally stacked) bars are connected by lines. The appearance of the lines can be controlled by the LineStyle() function. The LineStyle() function is explained in detail in the *Styles* section. Please note that the BarChartOptions() function should be listed after the BarChart() function. This rule generally applies: *chart options should always be listed after the actual chart function.*

Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 12 18 9 10;
            12 17 14 11 13 12;
            9 10 11 8 12 3)
  BarChart(proportional)
  BarChartOptions(on)
  FillStyle(3;darkYellow)
  LineStyle(all;poly;1;darkGray)
  BorderStyle(all;none)
  GridLocation(all;none)
  Scaling(y;percent)
CloseDrawing()

```

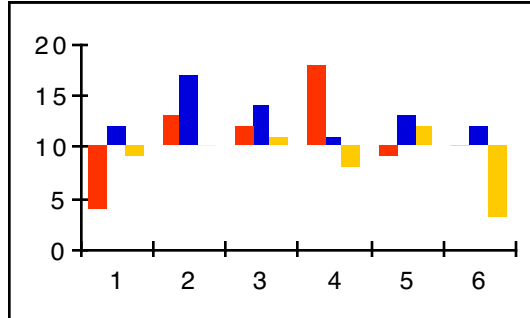


The 2nd argument *referenceValue* makes it possible to define the base line of the bars. As the default, all bars start at zero, i.e. *referenceValue=0*.  
Example:

```

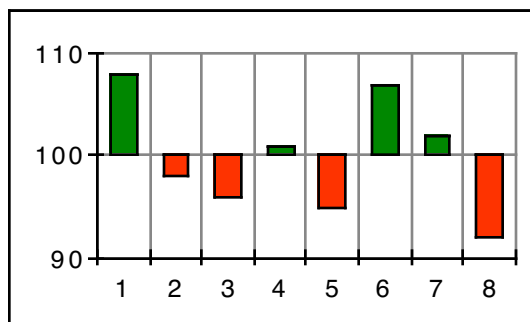
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 12 18 9 10;
            12 17 14 11 13 12;
            9 10 11 8 12 3)
  BarChart()
  BarChartOptions(;10) // reference value = 10
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
  GridLocation(all;none)
CloseDrawing()

```



By activating the 3rd argument *makeColorSplit=on*, the positive and negative values are represented by different colored bars. The appearance of the "positive bars", i.e. the chart values are greater than the reference value, can be controlled by using style functions with odd series indices; the appearance of "negative bars" by using style functions with even series indices. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(108 98 96 101 95 107 102 92)
  BarChart()
  BarChartOptions(;100;on) // reference value = 100
  FillStyle(1;green) // "positive bars", odd series index
  FillStyle(2;red) // "negative bars", even series index
CloseDrawing()
```

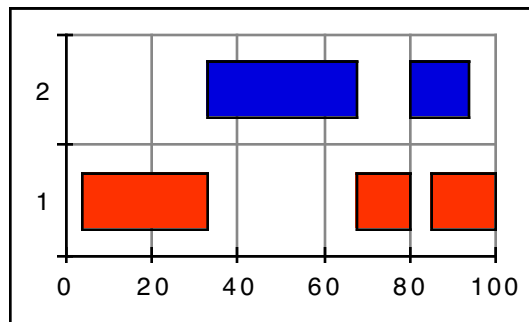


Please note that the color split for the legend is not supported.

**Gantt charts:****GanttChart (appearanceConstants;categoryGap;barDepth)**

The GanttChart() function serves to graphically organize time-dependent tasks, such as for project management. This type of chart was used for the first time in 1917 by Henry L. Gantt. Each task is defined by entering at least two values in ChartData(); the first value defines the beginning of a task, the second value the end. By defining more than two values per series, a task can be divided into several partial tasks. Example:

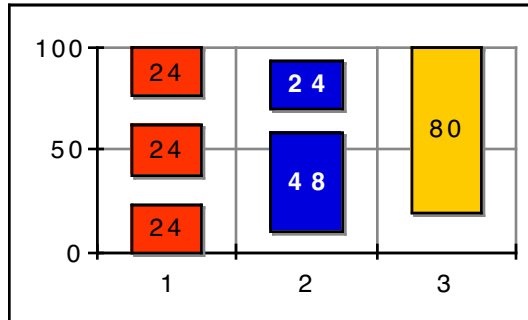
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 33 68 80 85 100; // task 1 (3 part. tasks)
            33 68 80 94)        // task 2 (2 part. tasks)
  GanttChart()
CloseDrawing()
```



The 1st argument *appearanceConstants* makes it possible to rotate the chart 90 degrees (*appearanceConstants=horizontal*) and add shadow (*appearanceConstants=shadow*) and labels (*appearanceConstants=label*) to the bars. All options can be combined by using a plus sign "+". Example:

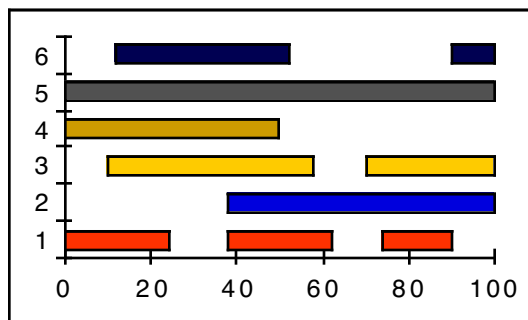
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 0 24 38 62 76 100;
            10 58 70 94;
            20 100)
  GanttChart(label+shadow+horizontal)
  FillStyle(3;darkYellow)
  LabelStyle(2;;;bold;white)
  ShadowStyle(all;1;gray)
```

CloseDrawing()



The fills, borders and shadows can be varied by using the style functions `FillStyle()`, `PictureStyle()`, `BorderStyle()` and `ShadowStyle()` and the labels by using the four style functions `LabelTexts()`, `LabelStyle()`, `LabelBackground()` and `LabelOptions()`. All style functions are discussed in detail in the *Styles* section. Examples:

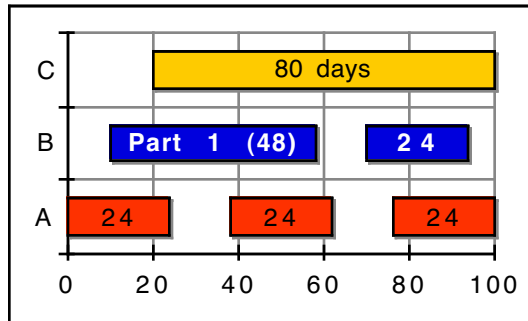
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 0 24 38 62 74 90;
             38 100;
             10 58 70 100;
             0 50;
             0 100;
             12 52 90 100)
  GanttChart()
  FillStyle(3;darkYellow)
  GridLocation(all;none)
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 0  24  38 62  76 100;
            10  58  70 94;
            20 100)
  GanttChart(label+shadow)
  FillStyle(3;darkYellow)
  LabelStyle(2;;;bold;white)
  LabelTexts(2;"Part 1 (|u|)";"|u|")
  LabelTexts(3;"|u| days")
  ShadowStyle(all;1;gray)
  AxisMajorTickLabelTexts(y;"A";"B";"C")
CloseDrawing()

```



By using the 2nd argument *categoryGap*, the space between the bars can be controlled. The argument *categoryGap* should be entered in percent of the bar width. As the default, the space between two bars is exactly one bar width, i.e. *categoryGap*=100. For a *categoryGap* less than 100 the space is smaller and the bars wider; for a *categoryGap* greater than 100 the space is bigger and the bars narrower.

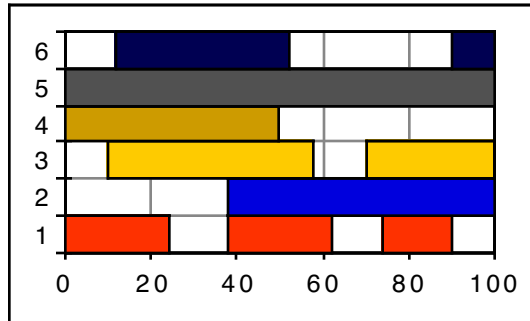
Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 0  24  38 62  74 90;
            38 100;
            10  58  70 100;
            0  50;
            0 100;
            12  52  90 100)
  GanttChart(;0)
  FillStyle(3;darkYellow)
  GridFrame()
CloseDrawing()

```

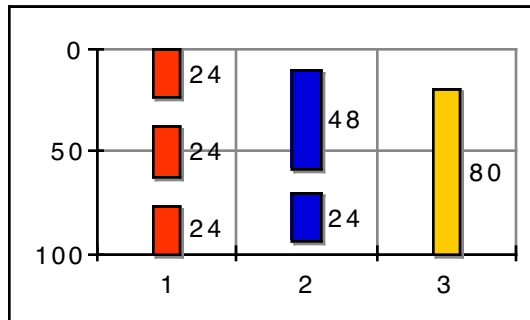




```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 0 24 38 62 76 100;
            10 58 70 94;
            20 100)
  GanttChart(label+shadow+horizontal;400)
  FillStyle(3;darkYellow)
  ShadowStyle(all;1;gray)
  ScalingOptions(y;on) // y-scaling top to bottom
CloseDrawing()

```

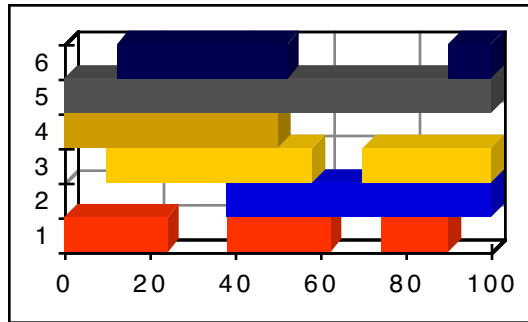


The 3rd argument *barDepth* can be used to control the three-dimensional drawing of bars. The bar depth is to be entered in percentage of the bar width. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 0 24 38 62 74 90;
            38 100; 10 58 70 100;
            0 50; 0 100;
            12 52 90 100)
  GanttChart(;0;50)
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
  GridFrame()
CloseDrawing()

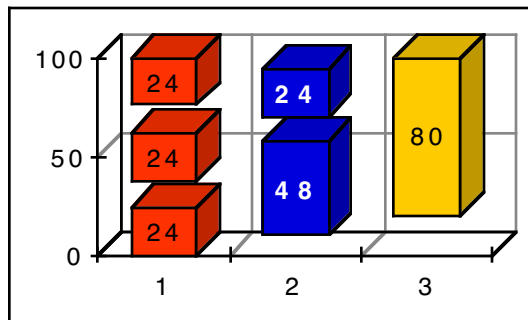
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 0 24 38 62 76 100;
            10 58 70 94; 20 100)
  GanttChart(label+horizontal;;50)
  FillStyle(3;darkYellow)
  LabelStyle(2;;;bold;white)
  ShadowStyle(all;1;gray)
CloseDrawing()

```



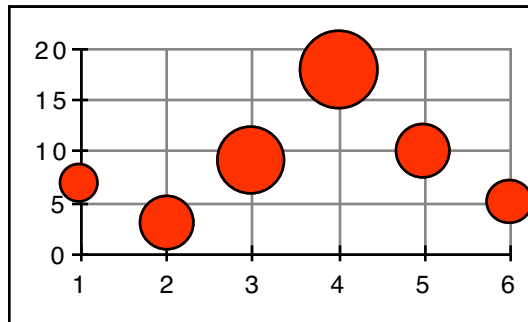
**Bubble charts:**

Bubble charts can be drawn both as one and two-dimensional.

**BubbleChart(*appearanceConstants*;doShiftIntervals)**

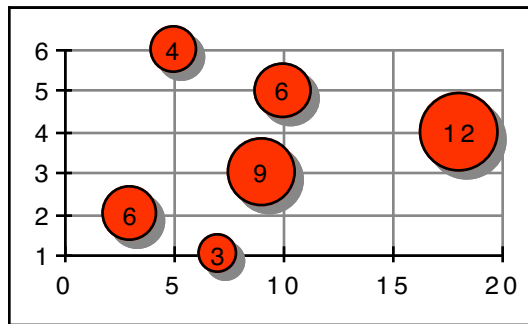
The BubbleChart() function serves to draw one-dimensional bubble charts. The 1st data series in the ChartData() function defines the positions, the 2nd data series the diameters of the 1st series, the 3rd and 4th data series in ChartData() the positions and diameters of the 2nd series, etc. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 3 9 18 10 5; // y-values
           3 6 9 12 6 4) // diameters
  BubbleChart()
CloseDrawing()
```



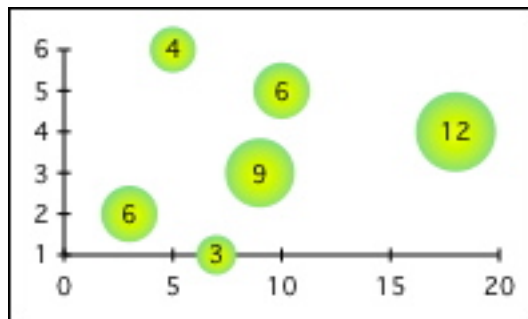
The 1st argument *appearanceConstants* makes it possible to rotate the chart 90 degrees (*appearanceConstants=horizontal*) and to add symbols (*appearanceConstants=symbol*), shadow (*appearanceConstants=shadow*) and labels (*appearanceConstants=label*) to the bubbles. All options can be combined by using a plus sign "+". Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 3 9 18 10 5; // x-values
           3 6 9 12 6 4) // diameters
  BubbleChart(horizontal+shadow+label)
CloseDrawing()
```



The fills, borders, symbols and shadows can be varied by using the style functions `FillStyle()`, `PictureStyle()`, `BorderStyle()`, `SymbolStyle()` and `ShadowStyle()` and the labels by using the four style functions `LabelTexts()`, `LabelStyle()`, `LabelBackground()` and `LabelOptions()`. All style functions are discussed in detail in the *Styles* section. Example:

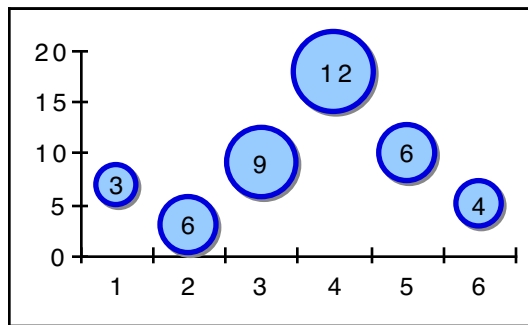
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 3 9 18 10 5; // x-values
            3 6 9 12 6 4) // diameters
  BubbleChart(label+horizontal)
  PictureStyle(1;resource;"39")
  BorderStyle(1;none)
  GridLocation(all;none)
CloseDrawing()
```



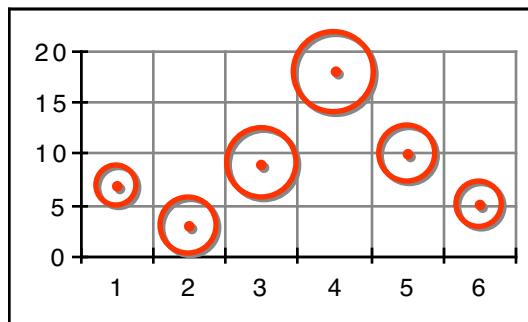
For one-dimensional bubble charts it is possible to move all bubbles half an interval width so that they do not lie on the interval borders but rather in the middle of the intervals. This can be done by activating the 2nd argument *doShiftIntervals=on*.

Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 3 9 18 10 5; // y-values
            3 6 9 12 6 4) // diameters
  BubbleChart(shadow+label;on)
  FillStyle(1;lightBlue)
  BorderStyle(1;;2;blue)
  ShadowStyle(all;1;gray)
  GridLocation(all;none)
CloseDrawing()
```



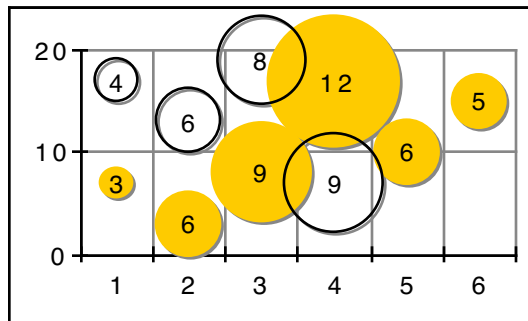
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 3 9 18 10 5; // y-values
            3 6 9 12 6 4) // diameters
  BubbleChart(shadow+symbol;on)
  FillStyle(1;;transparent)
  BorderStyle(1;;2;red)
  SymbolStyle(1;bullet;3;;red)
  ShadowStyle(all;1;gray)
CloseDrawing()
```



**BubbleChartOptions(maxDiameter;bubbleType)**

By using the 1st argument *maxDiameter*, the maximum bubble diameter can be controlled. If no value is defined, the maximum default bubble diameter equals 30 pixels. In addition, the proportion of the bubbles to each other can be controlled, i.e. when *bubbleType=areaProp*, the area of the bubbles are in proportion to each other; when *bubbleType=diameterProp*, the diameters are in proportion. The default is *bubbleType=areaProp*. Example:

```
OpenDrawing(200;120)
AddFrame(0;0;200;120)
ChartData(17 13 19 7;          // 1st series (y-values)
          4 6 8 9;            // 1st series (diameters)
          7 3 8 17 10 15;     // 2nd series (y-values)
          3 6 9 12 6 5)      // 2nd series (diameters)
BubbleChart(shadow+label;on)
BubbleChartOptions(50;diameterProp)
FillStyle(1;;transparent)
FillStyle(2;darkYellow)
BorderStyle(2;none)
ShadowStyle(all;1;gray)
CloseDrawing()
```



Please note that the `BubbleChartOptions()` function should be listed after the `BubbleChart()` or `BubbleChart2D()` function.

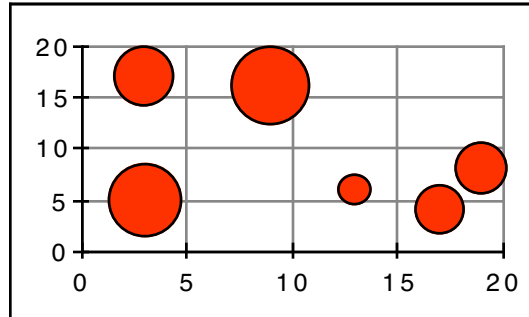
**BubbleChart2D(appearanceConstants)**

The `BubbleChart2D()` function makes it possible to draw two-dimensional bubble charts. The 1st data series in the `ChartData()` function defines the x-values, the 2nd data series the y-values and the 3rd data series the diameters of the first series, the 4th and 5th data series in `ChartData()` the x and y-values of the second series, the 6th data series the diameters of the second series, etc. Example:

```

OpenDrawing(200;120)
AddFrame(0;0;200;120)
ChartData(17 13 19 9 3 3; // x-value
          4 6 8 16 17 5; // y-values
          7 3 8 17 10 15) // diameters
BubbleChart2D()
CloseDrawing()

```

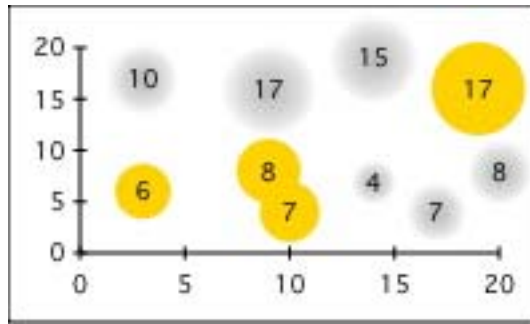


As for one-dimensional bubble charts, symbols (*appearanceConstants=*symbol), shadow (*appearanceConstants=*shadow) and labels (*appearanceConstants=*label) can be added to the bubbles by using the argument *appearanceConstants*. The options can be combined by using a plus sign "+". Rotating (*appearanceConstants=*horizontal) is not supported for two-dimensional charts as this is simply possible by switching the data series in *ChartData()*. Example:

```

OpenDrawing(200;120)
AddFrame(0;0;200;120)
ChartData(17 14 20 9 3 14; // 1st series: x-values
          4 7 8 16 17 19; // 1st series: y-values
          7 4 8 17 10 15; // 1st series: diameters
          10 3 9 19;      // 2nd series: x-values
          4 6 8 16;       // 2nd series: y-values
          7 6 8 17)       // 2nd series: diameters
BubbleChart2D(label)
BubbleChartOptions(35;areaProp)
PictureStyle(1;resource;"33")
FillStyle(2;darkYellow)
BorderStyle(all;none)
GridLocation(all;none)
CloseDrawing()

```

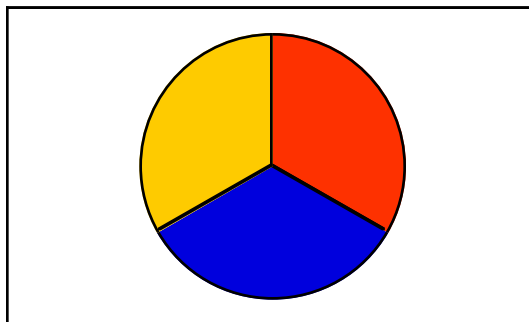


### Pie charts:

**PieChart(appearanceConstants;pieDepth;innerRadius;  
startAngle;arcAngle)**

The `PieChart()` function makes it possible to draw a variety of different pie charts. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(135 135 135)
  PieChart()
  FillStyle(3;darkYellow)
CloseDrawing()
```

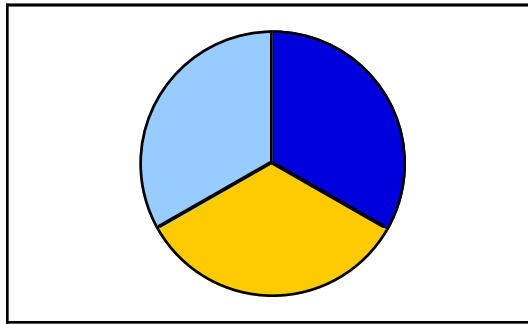


Only the first data series in `ChartData()` is represented; other data series are ignored. The same goes for values less than or equal to zero.



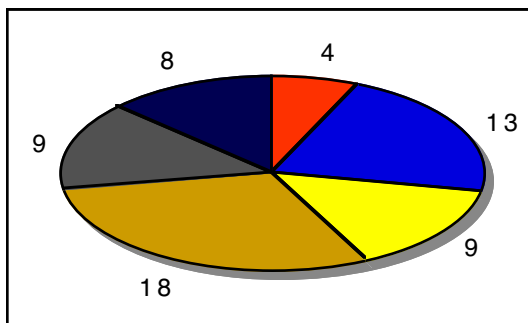
Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  // values less or equal 0 are skipped.
  ChartData(-100 135 135 0 135;
            100 200 300) // 2nd series is ignored
  PieChart()
  FillStyle(3;darkYellow)
  FillStyle(5;lightBlue)
CloseDrawing()
```



The 1st argument *appearanceConstants* makes it possible to choose between an elliptical form (*appearanceConstants=oval*), i.e. the entire available area can be made use of, or a circular shape. As the default, all pie charts are drawn circular. In addition, shadow (*appearanceConstants=shadow*) and labels (*appearanceConstants=label*) can be added to pie charts. All options can be combined by using a plus sign "+". Example:

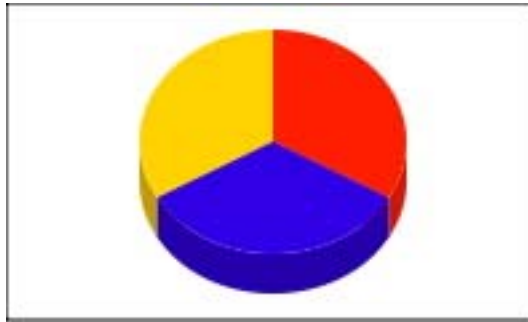
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 13 9 18 9 8)
  PieChart(oval+label+shadow)
CloseDrawing()
```



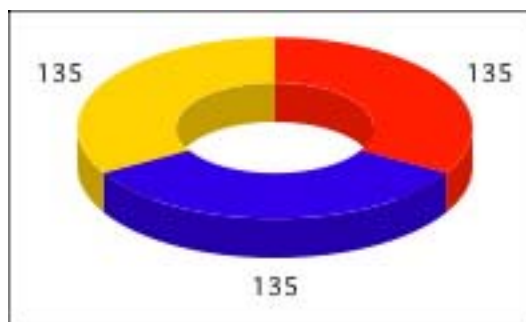
The fills, borders and shadows can be varied by using the style functions `FillStyle()`, `PictureStyle()`, `BorderStyle()` and `ShadowStyle()` and the labels by using the four style functions `LabelTexts()`, `LabelStyle()`, `LabelBackground()` and `LabelOptions()`. All style functions are discussed in detail in the *Styles* section.

The 2nd argument *pieDepth* makes it possible to draw three-dimensional pie charts. The pie depth should be entered in percent of the segment length (=pie chart radius). Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(135 135 135)
  PieChart(;30)
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
CloseDrawing()
```

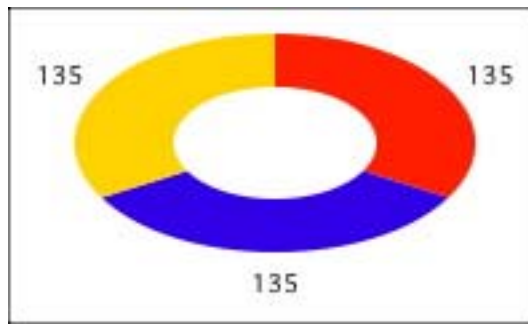


```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(135 135 135)
  PieChart(label+oval;20;50)
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
CloseDrawing()
```



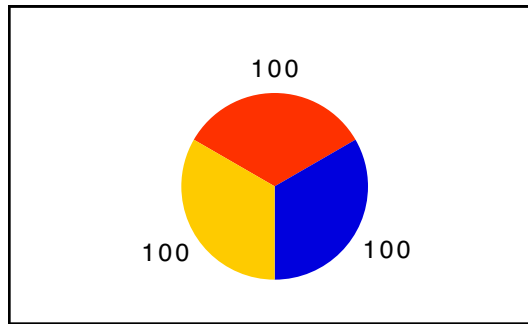
The 3rd argument *innerRadius* serves to draw ring charts. The inner radius should be entered in percent of the segment length (=pie chart radius).

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(135 135 135)
  PieChart(label+oval;;50)
  FillStyle(3;darkYellow)
  BorderStyle(all;none) // hide borders
CloseDrawing()
```



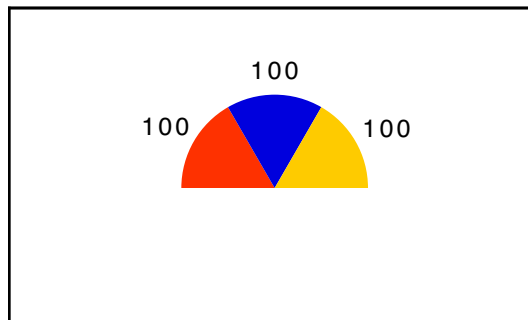
As the default, the segments of a pie chart are arranged clockwise over 360° (full circle), starting at the top (12 o'clock). By using the arguments *startAngle* and *arcAngle*, the position of the first segment as well as the direction and range (arc angle) of the pie chart can be defined. By defining a start angle greater than zero all segments are moved clockwise; by entering a negative start angle they are moved counterclockwise. All angles are to be entered within the range of ±360 degrees. A start angle of 0 degrees is at the top (12 o'clock), of 90 degrees is on the right (3 o'clock) and of -90 degrees is on the left (9 o'clock). Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(100 100 100)
  PieChart(label;;;-60)
  FillStyle(3;darkYellow)
  BorderStyle(all;none) // hide borders
CloseDrawing()
```

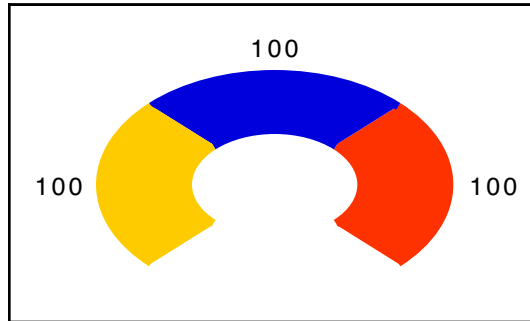


With an arc angle unequal to  $\pm 360$  degrees the segments are arranged in the form of a sector. A negative arc angle causes the segments to be arranged counterclockwise. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(100 100 100)
  PieChart(label;;;-90;180)
  FillStyle(3;darkYellow)
  BorderStyle(all;none) // hide borders
CloseDrawing()
```



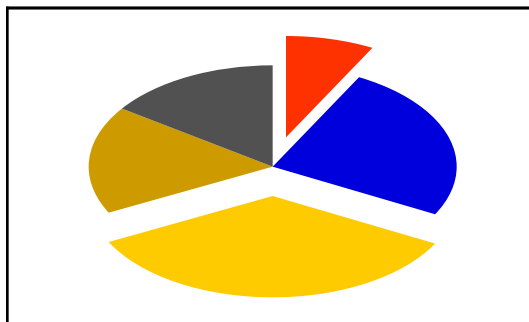
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(100 100 100)
  PieChart(oval+label;;50;135;-270)
  FillStyle(3;darkYellow)
  BorderStyle(all;none) // hide borders
CloseDrawing()
```



**PieChartExplodes(*explodeOffset*;*sliceIndex1*;  
*sliceIndex2*...)**

By using the `PieChartExplodes()` function, segments of a pie chart can be offset (exploding). The 1st argument *explodeOffset* defines the size of the offset value which should be entered in percent of the segment length. If no offset value is defined, the segments are offset by 20% of the segment length (*explodeOffset*=20). The other arguments in `PieChartExplodes()` define which segments should be offset. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 13 18 9 8)
  PieChart(oval)
  PieChartExplodes(30;1;3)
  FillStyle(3;darkYellow)
  BorderStyle(all;none) // hide borders
CloseDrawing()
```

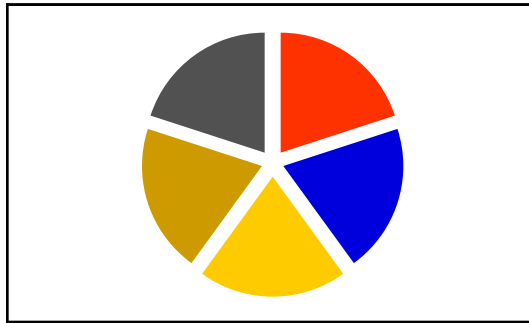


In addition, the following segment constants are also available:

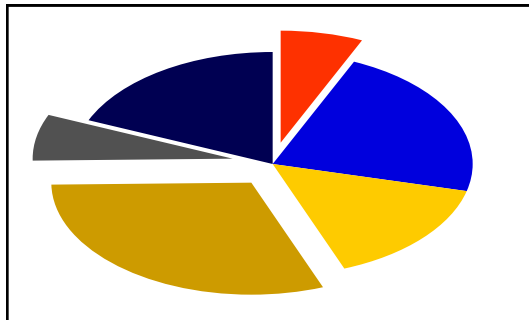
<i>constant</i>	<i>value</i>	<i>offset</i>
none	0	no segment
all	-1	all segments
max	-2	largest segment(s)
min	-3	smallest segment(s)

Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(72 72 72 72 72)
  PieChart()
  PieChartExplodes(10;all) // after PieChart()!
  FillStyle(3;darkYellow)
  BorderStyle(all;none) // hide borders
CloseDrawing()
```



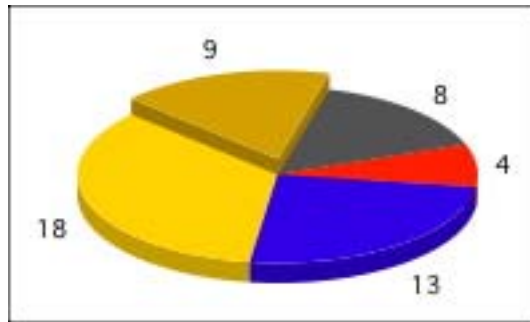
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 13 9 18 4 11)
  PieChart(oval)
  PieChartExplodes(;max;min) // after PieChart()!
  FillStyle(3;darkYellow)
  BorderStyle(all;none) // hide borders
CloseDrawing()
```



**PieChartExplodeDepths(explodeOffset;sliceIndex1;  
sliceIndex2...)**

By using the function `PieChartExplodeDepths()`, segments of a 3D pie chart can be moved upwards vertically. For 2D pie charts the function `PieChartExplodeDepths()` is ignored. The 1st argument *explodeOffset* defines the size of the offset value which should be entered in percent of the segment height. If no offset value is defined, the segments are offset by 20% of the segment height (*explodeOffset=20*). The other arguments in `PieChartExplodeDepths()` define which segments should be offset. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 13 18 9 8)
  PieChart(label+oval;10;;70)
  PieChartExplodeDepths(80;4) // after PieChart()!
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
CloseDrawing()
```



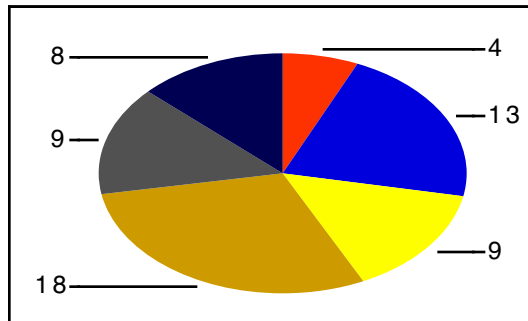
The functions `PieChartExplodes()` and `PieChartExplodeDepths()` can be entered several times and combined as well. Therefore, it is possible to move segments outwards radially and/or upwards vertically, just as you wish.

Please note that the functions `PieChartExplodes()` and `PieChartExplodeDepths()` should be listed after the `PieChart()` function. This rule generally applies: *chart options should always be listed after the actual chart function.*

**PieChartAuxLines(*horizontalLength*;*extensionLength*;  
vAlignment;*width*;*color*;*pattern*)**

The PieChartAuxLines() function can be used to better position the outside labels. Example:

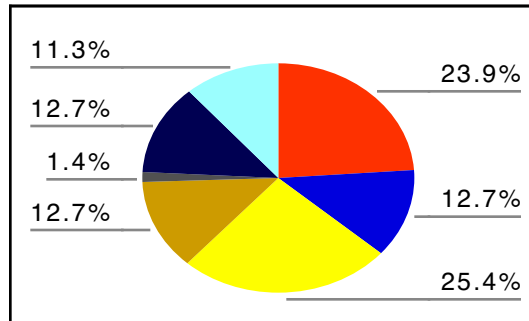
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 13 9 18 9 8)
  PieChart(label+oval)
  PieChartAuxLines() // after PieChart()!
  BorderStyle(all;none) // hide borders
CloseDrawing()
```



The length of the auxiliary lines can be controlled by the argument *horizontalLength*. It is to be entered in percent of the segment length. The argument *extensionLength* is currently not used and is ignored. By using the argument *vAlignment*, the position of the auxiliary line can be defined relative to the label. When *vAlignment*=*bottom*, the auxiliary line is positioned below the label, when *vAlignment*=*top* above the label, when *vAlignment*=*center* in the center (default). The line width, color and pattern can be varied by using the arguments *width*, *color* and *pattern*. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(17 9 18 9 1 9 8)
  PieChart(label+oval)
  PieChartAuxLines(;;bottom;1;gray) // after PieChart()!
  LabelTexts("|||f1|")
  BorderStyle(all;none) // hide borders
CloseDrawing()
```

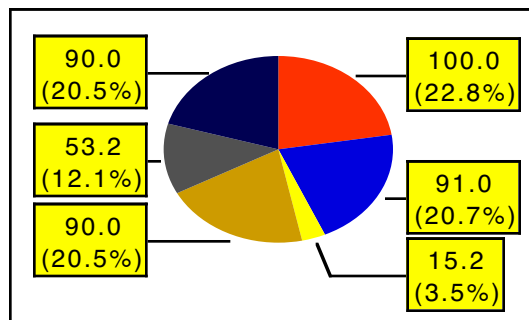




```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(100 91 15.2 90 53.2 89.97)
  PieChart(label+oval)
  PieChartAuxLines(;20) // after PieChart(!)
  LabelTexts(";|f1|\n(|f1|%)")
  LabelStyle(;;;;center)
  LabelBackground(1;yellow)
  BorderStyle(all;none) // hide borders
CloseDrawing()

```



The formatted output of the absolute and/or relative values is explained in detail together with examples in the *Styles* section.

**PieChartLabelOptions(useRelativeLimits;  
outerLabelOffset;innerLabelOffset)**

The 1st argument *useRelativeLimits* makes it possible to define relative limits instead of absolute limits. Chart values, which do not lie within these limits, are not labeled. The limits, i.e. the lower and upper limit, are defined in the `LabelOptions()` function. By using the arguments *outerLabelOffset* and *innerLabelOffset*, the space between the border and the outer and inner labels can be controlled. Examples can be found in the *Styles* section.

```

PieChartInnerLabelTexts(text1;text2...)
PieChartInnerLabelStyle(font;size;style;color;
    alignment;orientation;maxWidth;
    maxHeight;ellipsisPosition)
PieChartInnerLabelBackground(fillColor;fillPattern;
    borderWidth;borderColor;borderPattern;
    shadowOffset;shadowColor;shadowPattern)

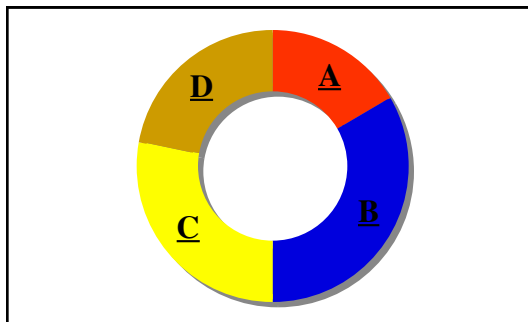
```

As the default, labels are positioned outside the pie chart (*appearance-Constants=label*). By using the `PieChartInnerLabelTexts()`, `PieChartInnerLabelStyle()` and `PieChartInnerLabelBackground()` functions, it is also possible to place labels inside the chart. The setup and operation of the three functions is the same as for the "standard" `LabelTexts()`, `LabelStyle()` and `LabelBackground()` functions. The latter is dealt with in detail in the *Styles* section. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(45 90 75 60)
  PieChart(shadow;;60)
  PieChartInnerLabelTexts("A";"B";"C";"D")
  PieChartInnerLabelStyle("Times";12;bold+underline)
  BorderStyle(all;none) // hide borders
  ShadowStyle(all;2;gray)
CloseDrawing()

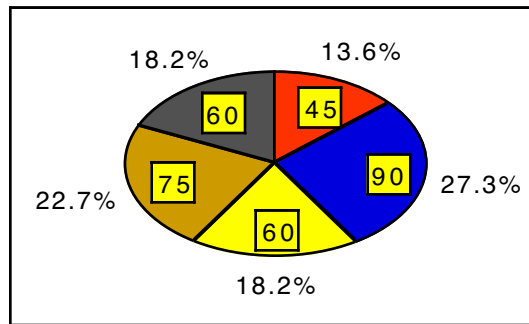
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(45 90 60 75 60)
  PieChart(label+oval)
  LabelTexts(1;"|||f1|%" ) // relative values
  PieChartInnerLabelTexts("|u|") // absolute values
  PieChartInnerLabelBackground(yellow)
CloseDrawing()

```



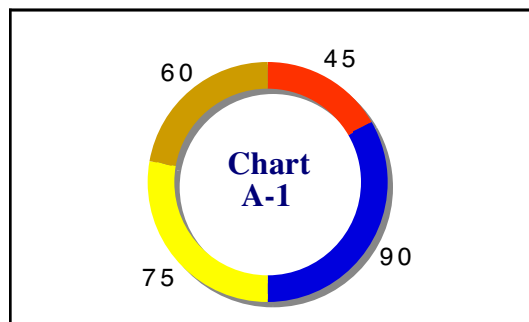
**PieChartCenterLabelText(text)**

**PieChartCenterLabelStyle(font;size;style;color;  
alignment;orientation;maxWidth;  
maxHeight;ellipsisPosition)**

**PieChartCenterLabelBackground(fillColor;fillPattern;  
borderWidth;borderColor;borderPattern;  
shadowOffset;shadowColor;shadowPattern)**

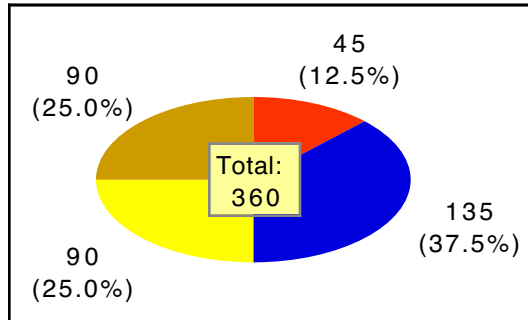
In addition, by using the `PieChartCenterLabelText()`, `PieChartCenterLabelStyle()` and `PieChartCenterLabelBackground()` functions, a text can be placed and drawn in the center of the chart. The three functions work the same way as the "standard" `LabelTexts()`, `LabelStyle()` and `LabelBackground()` functions; see *Styles* section. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(45 90 75 60)
  PieChart(shadow+label;;85)
  PieChartCenterLabelText("Chart\nA-1")
  PieChartCenterLabelStyle("Times";12;bold;darkBlue)
  BorderStyle(all;none) // hide borders
  ShadowStyle(all;2;gray)
CloseDrawing()
```



If the text in the `PieChartCenterLabelText()` function contains a format specifier, e.g. "`|u|`", the sum of all positive values is shown. A detailed explanation of all format specifiers, including numerous examples, can be found in *xmReference*. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(45 135 90 90)
  PieChart(oval+label)
  LabelTexts(1;"|u|\n(|f1|%)")
  LabelStyle(;;;;center)
  PieChartCenterLabelText("Total: \n|u|")
  PieChartCenterLabelBackground(lightYellow;;1;gray)
  BorderStyle(all;none) // hide borders
CloseDrawing()
```

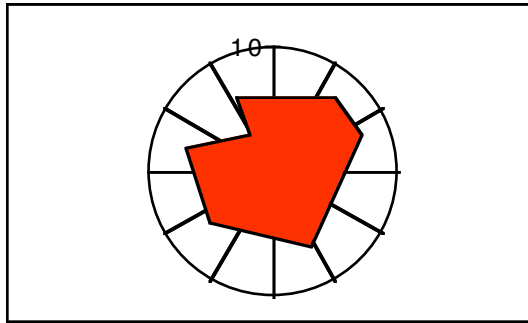


### Polar charts:

#### **PolarChart(appearanceConstants;startAngle;arcAngle)**

The `PolarChart()` function makes it possible to draw two-dimensional values in polar coordinates. The 1st data series in the `ChartData()` function defines the x-values, the 2nd data series the y-values of the first series, the 3rd and 4th data series in `ChartData()` the x and y-values of the second series, etc. Example:

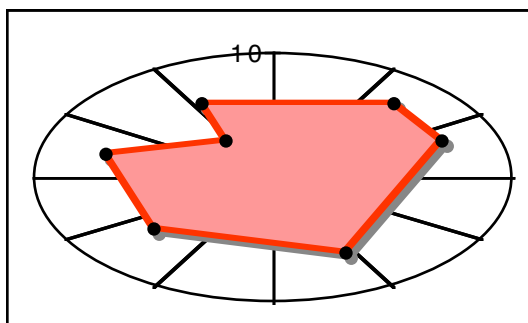
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 7 3 -5 -7 -2 -3; // x-values
           6 3 -6 -4 2 3 6) // y-values
  PolarChart()
  MajorGridLineColors(all;all;black)
CloseDrawing()
```



The 1st argument *appearanceConstants* makes it possible to choose between an elliptical form (*appearanceConstants=oval*), i.e. the entire available area can be made use of, or a circular shape. As the default, all polar charts are drawn circular. In addition, symbols (*appearanceConstants= symbol*), shadow (*appearanceConstants=shadow*) and labels (*appearanceConstants=label*) can be added to charts. All options can be combined by using a plus sign "+".

The fills, borders, symbols and shadows can be varied by using the style functions *FillStyle()*, *PictureStyle()*, *BorderStyle()*, *SymbolStyle()* and *ShadowStyle()* and the labels by using the four style functions *LabelTexts()*, *LabelStyle()*, *LabelBackground()* and *LabelOptions()*. All style functions are discussed in detail in the *Styles* section. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 7 3 -5 -7 -2 -3; // x-values
           6 3 -6 -4 2 3 6) // y-values
  PolarChart(oval+shadow+symbol)
  FillStyle(1;lightRed)
  BorderStyle(1;poly;2;red)
  SymbolStyle(1;bullet;4;1;black)
  ShadowStyle(1;2;gray)
  MajorGridLineColors(all;all;black)
CloseDrawing()
```



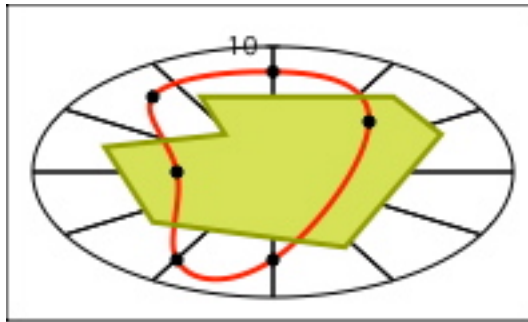
```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 7 3 -5 -7 -2 -3; // 1st series: x-values
            6 3 -6 -4 2 3 6; // 1st series: y-values
            0 4 0 -4 -4 -5; // 2nd series: x-values
            8 4 -7 -7 0 6) // 2nd series: y-values
  PolarChart(oval+symbol)

  // 1st series
  FillStyle(1;220 220 100)
  BorderStyle(1;poly;2;150 150 0)
  SymbolStyle(1;none)

  // 2nd series
  FillStyle(2;;transparent)
  BorderStyle(2;smooth;2;red)
  SymbolStyle(2;bullet;4;1;black)
  MajorGridLineColors(all;all;black)
CloseDrawing()

```

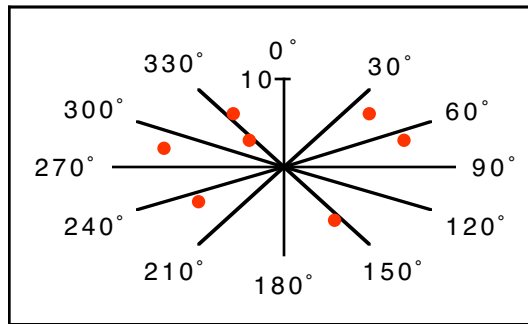


As the default, the points of a series are connected to a polygon. By suppressing both the fill and border using `FillStyle(;;transparent)` and `BorderStyle(;none)`, it is possible to represent the chart values solely by using symbols. Example:

```

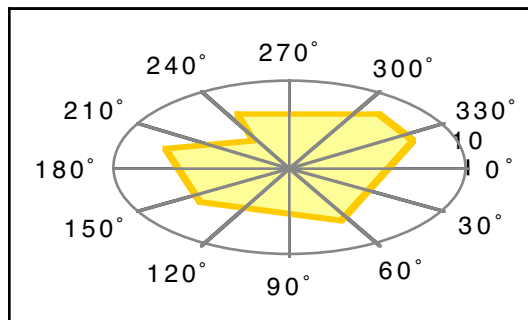
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 7 3 -5 -7 -2 -3; // x-values
            6 3 -6 -4 2 3 6) // y-values
  PolarChart(oval+symbol)
  FillStyle(;;transparent)
  BorderStyle(;none)
  SymbolStyle(1;bullet;4;1;red)
  AxisLabelText(all;"|u|°")
  GridLocation(all;none) // hide grid
CloseDrawing()

```



The texts for the axis labels are entered by using the `AxisLabelText()` function. Details can be found in the *Axes* section. As the default, the axis labels start at the top (12 o'clock) and run clockwise over a range of 360°. By using the arguments *startAngle* and *arcAngle*, the position of the 0 degrees line as well as the direction and range (arc angle) of the grid can be defined. By defining a start angle greater than zero all axes are moved clockwise; by entering a negative start angle they are moved counterclockwise. All angles are to be entered within the range of  $\pm 360$  degrees. A start angle of 0 degrees is at the top (12 o'clock), of 90 degrees is on the right (3 o'clock) and of -90 degrees is on the left (9 o'clock). Examples:

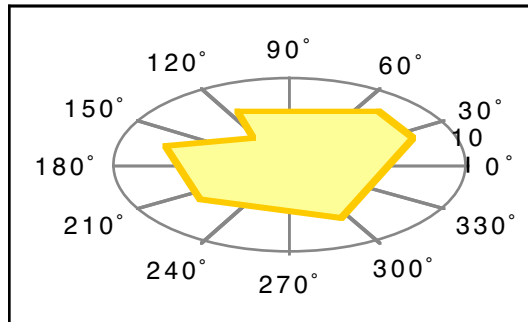
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 7 3 -5 -7 -2 -3; // x-values
            6 3 -6 -4 2 3 6) // y-values
  PolarChart(oval;90)
  FillStyle(1;lightYellow)
  BorderStyle(1;poly;2;darkYellow)
  GridLocation(all;front)
  AxisLabelText(all;"|u|°")
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 7 3 -5 -7 -2 -3; // x-values
            6 3 -6 -4 2 3 6) // y-values
  // 90...0-degrees-line on the right.
  // -360...degrees run counterclockwise.
  PolarChart(oval;90;-360)
  FillStyle(1;lightYellow)
  BorderStyle(1;poly;2;darkYellow)
  AxisLine(1;1;gray)
  AxisLabelText(all;"|u|°")
CloseDrawing()

```



**PolarChartOptions(*scalingAxisIndex*;*gridShape*;  
doAddArrows;doNotClosePolygon;numOfAxes)**

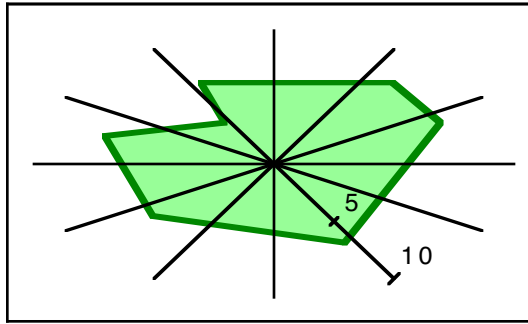
Polar charts can be varied in many ways by using the `PolarChartOptions()` function. Please note that the `PolarChartOptions()` function should be entered after the `PolarChart()` function. By using the 1st argument *scalingAxisIndex* it is possible to define along which axis the scaling will be shown. As the default, the scaling is placed along the first axis (*scalingAxisIndex=1*), this is the axis at 0 degrees. When *scalingAxisIndex=0*, no scaling is shown. Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 7 3 -5 -7 -2 -3; // x-values
            6 3 -6 -4 2 3 6) // y-values
  PolarChart(oval)
  PolarChartOptions(6) // after PolarChart()!
  AxisOptions(all;front)
  FillStyle(1;lightGreen)
  BorderStyle(1;poly;2;green)
  GridLocation(all;none) // hide grid
CloseDrawing()

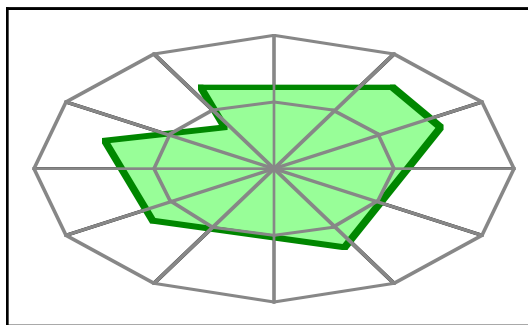
```





By using the 2nd argument *gridShape*, it is possible to choose between the default oval grid (*gridShape=oval*) and a polygonal grid (*gridShape=poly*) or to suppress the grid (*gridShape=none*). The grid can also be suppressed by using `GridLocation(;none)`. Example:

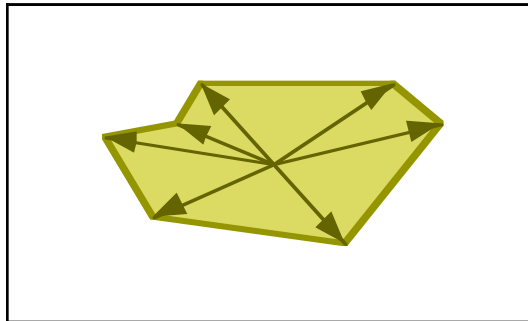
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 7 3 -5 -7 -2 -3; // x-values
            6 3 -6 -4 2 3 6) // y-values
  PolarChart(oval)
  PolarChartOptions(0;poly) // after PolarChart()!
  FillStyle(1;lightGreen)
  BorderStyle(1;poly;2;green)
  GridLocation(all;front)
CloseDrawing()
```



As an option, arrows originating from the center can be added to polar charts (*doAddArrows=on*). The appearance of the arrows can be controlled by using the `ArrowStyle()` function which is explained in the *Styles* section.

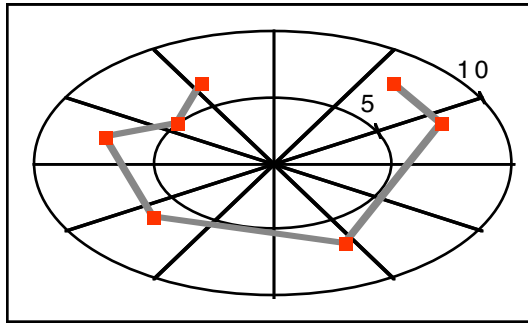
Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 7 3 -5 -7 -4 -3; // x-values
            6 3 -6 -4 2 3 6) // y-values
  PolarChart(oval)
  PolarChartOptions(0;none;on) // after PolarChart()!
  FillStyle(1;220 220 100)
  BorderStyle(1;poly;2;150 150 0)
  ArrowStyle(1;1;100 100 0;;;12;8)
  AxisOptions(all;none) // hide axes
CloseDrawing()
```



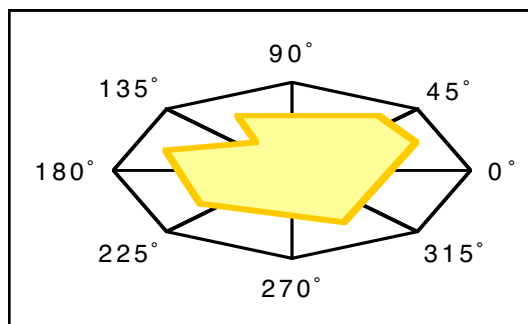
By using `FillStyle(;;transparent)`, the fill can be suppressed; the remaining border connects the points in the form of a closed line. By activating the 4th argument *doNotClosePolygon=on*, the closing line between the last and first point can be suppressed. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 7 3 -5 -7 -4 -3; // x-values
            6 3 -6 -4 2 3 6) // y-values
  PolarChart(oval+symbol)
  PolarChartOptions(3;oval;off;on) // after PolarChart()
  FillStyle(1;;transparent)
  BorderStyle(1;poly;2;gray)
  SymbolStyle(1;square;4;1;red)
  MajorGridLineColors(all;all;black)
CloseDrawing()
```



The number of radial grid lines (axes) can be controlled by using the 5th argument *numOfAxes*. As the default, polar charts are drawn with 12 radial grid lines. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 7 3 -5 -7 -2 -3; // x-values
           6 3 -6 -4 2 3 6) // y-values
  // 90...0-degrees-axis on the right.
  // -360...degrees run counterclockwise.
  PolarChart(oval;90;-360)
  PolarChartOptions(0;poly;off;;8) // after PolarChart()
  FillStyle(1;lightYellow)
  BorderStyle(1;poly;2;darkYellow)
  MajorGridLineColors(all;all;black)
  AxisLabelText(all;"|u|°")
CloseDrawing()
```

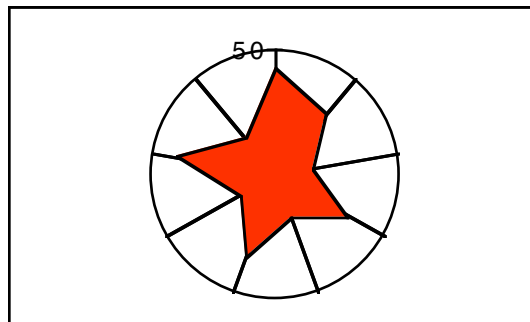


**Radar charts:**

**RadarChart (appearanceConstants;startAngle;arcAngle;  
doShiftIntervals)**

By using the RadarChart() function, chart values are placed along radial axes. The maximum number of values per data series determines the number of axes. As the default, the individual values of a data series are connected to a polygon. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(43 32 15 33 18 35 16 40 19)
  RadarChart()
  MajorGridLineColors(all;all;black)
CloseDrawing()
```



The 1st argument *appearanceConstants* makes it possible to choose between an elliptical form (*appearanceConstants=oval*), i.e. the entire available area can be made use of, or a circular shape. As the default, all polar charts are drawn circular. In addition, symbols (*appearanceConstants= symbol*), shadow (*appearanceConstants=shadow*) and labels (*appearanceConstants=label*) can be added to charts. All options can be combined by using a plus sign "+".

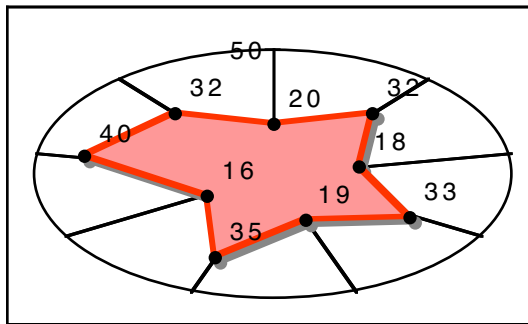
The fills, borders, symbols and shadows can be varied by using the style functions FillStyle(), PictureStyle(), BorderStyle(), SymbolStyle() and ShadowStyle() and the labels by using the four style functions LabelTexts(), LabelStyle(), LabelBackground() and LabelOptions(). All style functions are discussed in detail in the *Styles* section.

## Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(20 32 18 33 19 35 16 40 32)
  RadarChart(oval+symbol+label+shadow)
  FillStyle(1;lightRed)
  BorderStyle(1;poly;2;red)
  ShadowStyle(1;2;gray)
  SymbolStyle(1;bullet;4;;black)
  MajorGridLineColors(all;all;black)
CloseDrawing()

```



```

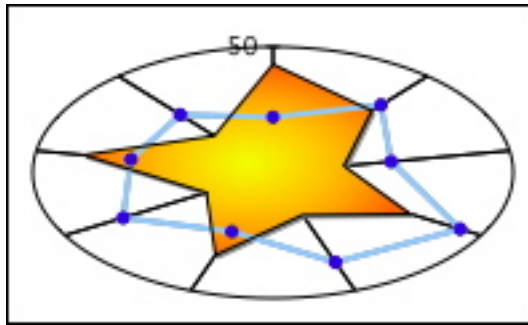
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(43 32 15 33 18 35 16 40 19; // 1st series
            22 35 25 45 38 25 36 30 30) // 2nd series
  RadarChart(oval+symbol+shadow)

  // 1st series
  PictureStyle(1;resource;"41")
  SymbolStyle(1;none)
  ShadowStyle(1;1;gray)

  // 2nd series
  FillStyle(2;;transparent)
  BorderStyle(2;poly;2;lightBlue)
  SymbolStyle(2;bullet;4;1;)
  ShadowStyle(2;0)

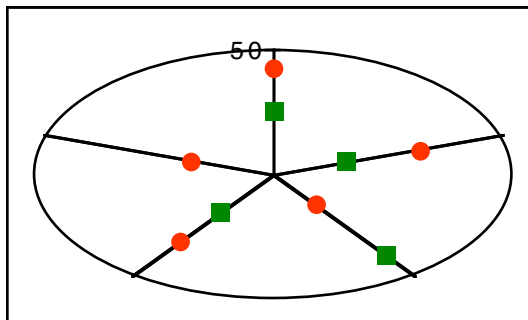
  MajorGridLineColors(all;all;black)
CloseDrawing()

```



By suppressing the fill and border, it is possible to depict chart values solely by using symbols. Example:

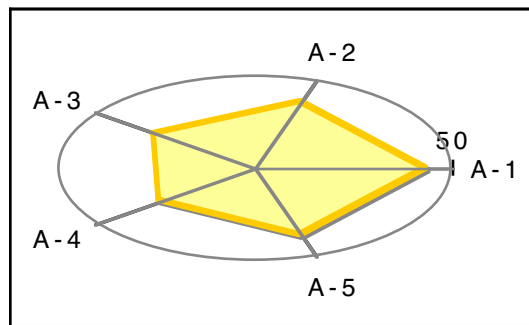
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(43 32 15 33 18; // 1st series
            25 16 40 19)   // 2nd series
  RadarChart(oval+symbol)
  FillStyle(;;transparent)
  BorderStyle(;;none)
  SymbolStyle(1;bullet;6;1;red)
  SymbolStyle(2;square;6;1;green)
  MajorGridLineColors(all;all;black)
CloseDrawing()
```



The texts for the axis labels are defined by using the `AxisLabelText()` function. Details can be found in the *Axes* section. On radar charts the axis index is always 1; axis indices greater than 1 are ignored. As the default, the axes start at the top (12 o'clock) and run clockwise over a range of 360°. By using the arguments *startAngle* and *arcAngle*, the position of the first axis as well as the direction and range (arc angle) of the axes can be defined. By defining a start angle greater than zero all axes are moved clockwise; by entering a negative start angle they are moved counterclockwise. All angles are to be entered within the range

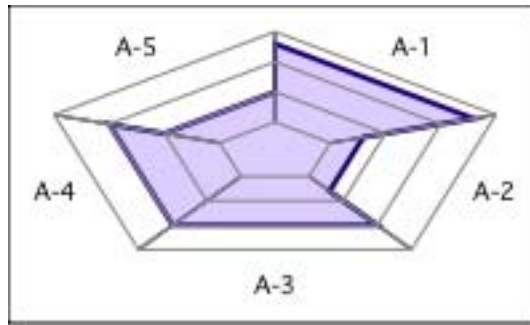
of  $\pm 360$  degrees; A start angle of 0 degrees is at the top (12 o'clock), of 90 degrees is on the right (3 o'clock) and of -90 degrees is on the left (9 o'clock). Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(43 38 32 30 38)
  RadarChart(oval+shadow;90;-360)
  FillStyle(1;lightYellow)
  BorderStyle(1;poly;2;darkYellow)
  ShadowStyle(1;1;gray)
  GridLocation(all;front)
  AxisLabelText(all;"A-|u|")
CloseDrawing()
```



By using the 4th argument *doShiftIntervals=on*, the values to be represented will not be drawn vertex-oriented as points along the radial axes but rather wedge-oriented between the axes. Example:

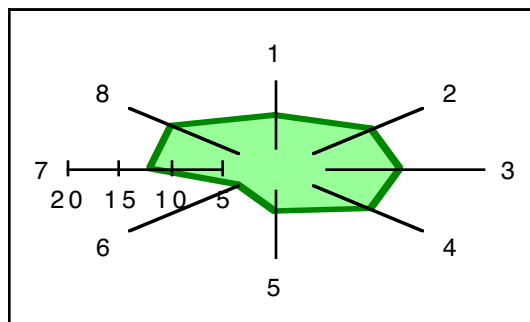
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(18 8 15 15 10)
  RadarChart(oval;;;on)
  RadarChartOptions(0;poly) // after RadarChart()!
  FillStyle(1;50 0 255 50)
  BorderStyle(1;poly;2;darkBlue)
  GridLocation(all;front)
  Scaling(x;linear;5;20;3)
  AxisLabelText(all;"A-|u|")
CloseDrawing()
```



**RadarChartOptions(*scalingAxisIndex*;*gridShape*;  
doAddArrows;doNotClosePolygon)**

Radar charts can be varied in many ways by using the RadarChartOptions() function. Please note that the RadarChartOptions() function should be entered after the RadarChart() function. By using the 1st argument *scalingAxisIndex* it is possible to define along which axis the scaling will be shown. As the default, the scaling is placed along the first axis (*scalingAxisIndex=1*). When *scalingAxisIndex=0*, no scaling is shown. Example:

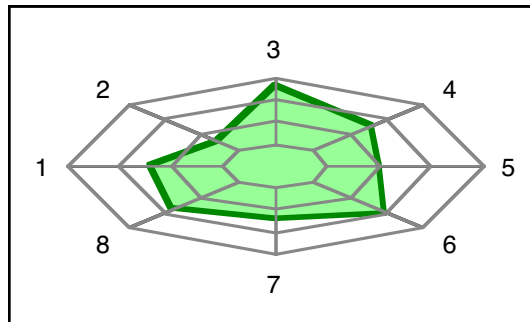
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(12 13 12 13 10 5 12 14)
  RadarChart(oval)
  RadarChartOptions(7) // after RadarChart()!
  AxisOptions(all;front)
  FillStyle(1;lightGreen)
  BorderStyle(1;poly;2;green)
  GridLocation(all;none) // hide grid
  Scaling(x;linear;5;20;3)
  AxisLabelText(all;"|u|")
CloseDrawing()
```



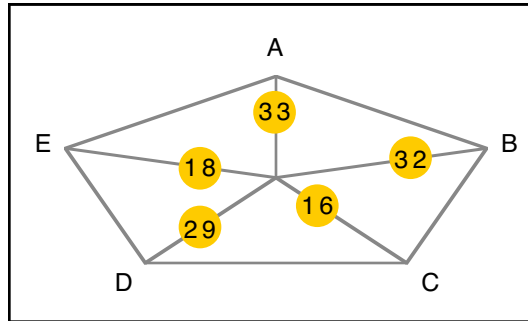


By using the 2nd argument *gridShape*, it is possible to choose between the default oval grid (*gridShape=oval*) and a polygonal grid (*gridShape=poly*) or to suppress the grid (*gridShape=none*). The grid can also be suppressed by using *GridLocation(;none)*. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(12 8 18 13 10 15 12 14)
  RadarChart(oval;-90)
  RadarChartOptions(0;poly) // after RadarChart()!
  FillStyle(1;lightGreen)
  BorderStyle(1;poly;2;green)
  GridLocation(all;front)
  Scaling(x;linear;5;20;3)
  AxisLabelText(all;"|u|")
CloseDrawing()
```

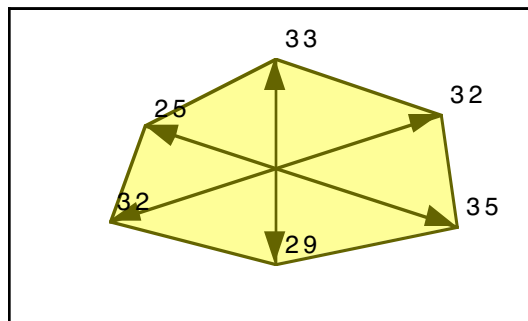


```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(33 32 16 29 18)
  RadarChart(oval+label+symbol)
  RadarChartOptions(0;poly) // after RadarChart()!
  FillStyle(1;;transparent)
  BorderStyle(1;none)
  SymbolStyle(1;bullet;15;;darkYellow)
  LabelOptions(1;centerCenter)
  AxisLine(1;0) // hide axis lines
  AxisLabelText(1;"A";"B";"C";"D";"E")
CloseDrawing()
```

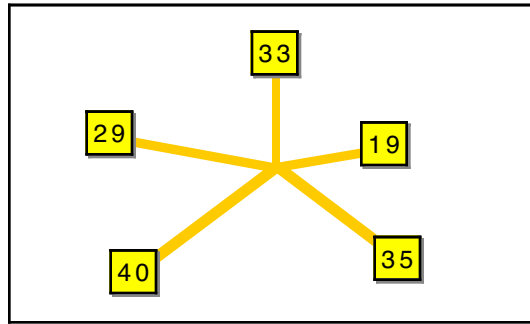


As an option, arrows originating from the center can be added to radar charts (*doAddArrows=on*). The appearance of the arrows can be controlled by the `ArrowStyle()` function which is explained in the *Styles* section. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(33 32 35 29 32 25)
  RadarChart(oval+label)
  RadarChartOptions(0;none;on) // after RadarChart()!
  FillStyle(1;lightYellow)
  BorderStyle(1;;1;100 100 0)
  ArrowStyle(1;1;100 100 0;;;12;8)
  AxisOptions(all;none) // hide axes
CloseDrawing()
```



```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(33 19 35 40 29)
  RadarChart(oval+label)
  RadarChartOptions(0;none;on) // after RadarChart()!
  FillStyle(1;;transparent)
  BorderStyle(1;none)
  LabelBackground(1;yellow;;;1;black;;1)
  LabelOptions(1;centerCenter)
  ArrowStyle(1;3;darkYellow;;;0;0)
  AxisOptions(all;none) // hide axes
CloseDrawing()
```

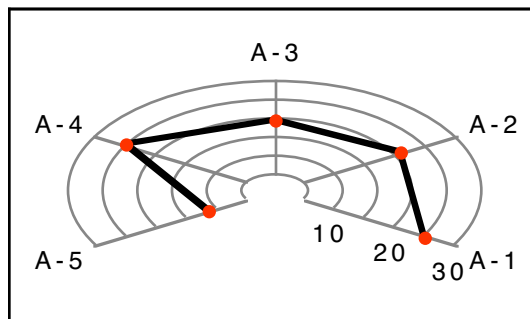


By using `FillStyle(;;transparent)`, the fill can be suppressed; the remaining border connects the points in the form of a closed line. By activating the 4th argument `doNotClosePolygon=on`, the closing line between the last and first point can be suppressed. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(25 21 19 25 11)
  RadarChart(symbol+oval;120;-240)
  RadarChartOptions(1;oval;off;on) // after RadarChart()

  FillStyle(all;;transparent)
  BorderStyle(1;poly;2;black)
  SymbolStyle(1;bullet;4;1;red)

  Scaling(1;linear;5;30;5)
  AxisLabelText(1;"A-|i0|")
  AxisLine(1;0) // hide axis lines
  AxisMajorTicks(1;0) // hide tick marks
  AxisMajorTickLabelOptions(1;in;;;2;2)
CloseDrawing()
```



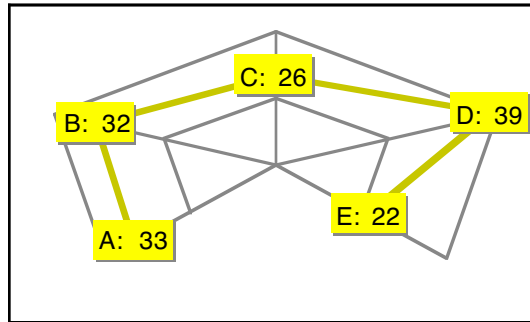
```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(33 32 26 39 22)
  RadarChart(oval+label;-135;270)
  RadarChartOptions(0;poly;off;on) // after RadarChart()
  LabelTexts(1;"A: |u|";"B: |u|";"C: |u|";
              "D: |u|";"E: |u|")
  LabelBackground(1;yellow;;0;;1)
  LabelOptions(1;centerCenter)

  FillStyle(1;;transparent)
  BorderStyle(1;;2;200 200 0)

  AxisOptions(all;none) // hide axes
CloseDrawing()

```



### High-Low charts:

High-Low charts can be drawn both as one and two-dimensional.

```

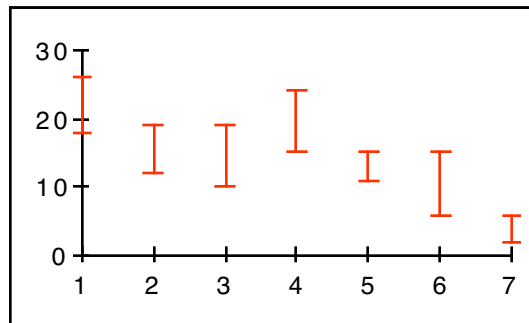
HighLowChart(appearanceConstants;doShiftIntervals;
             highLowChartType;highMarkerLength;
             highMarkerAlignment;lowMarkerLength;
             lowMarkerAlignment;closeMarkerLength;
             closeMarkerAlignment;openMarkerLength;
             openMarkerAlignment)

```

The HighLowChart() function makes it possible to depict value ranges in a variety of ways. At least two data series are necessary to define the range limits. The highest values are entered in the ChartData() function as the 1st data series, the lowest values as the 2nd data series.

Example:

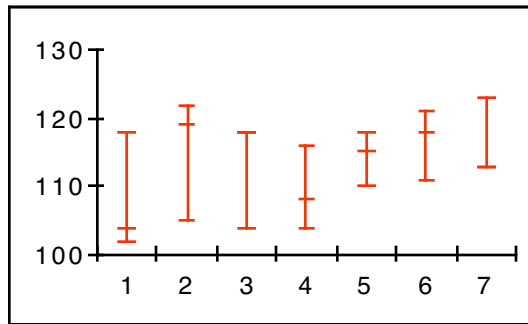
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(26 19 19 24 15 15 6; // high-values
            18 12 10 15 11 6 2) // low-values
  HighLowChart()
  GridLocation(all;none) // hide grid
CloseDrawing()
```



As an option, by using the 3rd argument *highLowChartType*, three values (*highLowChartType=highLowClose*) or even four values (*highLowChartType=highLowCloseOpen*) can be entered in addition to the two default values (*highLowChartType=highLow*). Since this type of chart is frequently used for stock market prices, the four values are referred to as highest, lowest, closing and opening price. In *ChartData()* the closing prices are entered as the 3rd data series and the opening prices as the 4th data series.

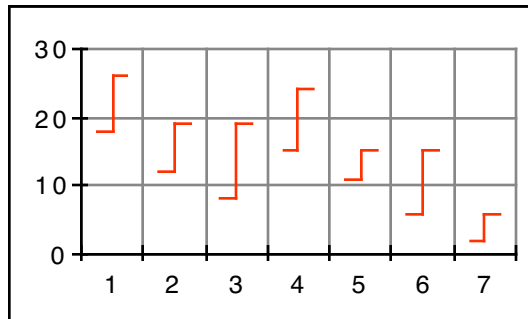
Furthermore, it is possible to move the range lines half an interval width so that they do not lie on the interval borders but rather in the middle of the intervals. This can be done by activating the 2nd argument *doShift-Intervals=on*. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(118 122 118 116 118 121 123; // high-values
            102 105 104 104 110 111 113; // low-values
            104 119 118 108 115 118 113) // close-values
  HighLowChart(;on;highLowClose)
  GridLocation(all;none) // hide grid
CloseDrawing()
```



The alignment of the markers — the markers for the lowest and opening prices are normally aligned to the left, the markers for the highest and closing prices to the right — is defined by using the arguments *high-MarkerAlignment*, *lowMarkerAlignment*, *closeMarkerAlignment* and *open-MarkerAlignment*. All markers can either be placed on the left, on the right or in the center (default). The length of the markers can be controlled by the arguments *highMarkerLength*, *lowMarkerLength*, *close-MarkerLength* and *openMarkerLength* and should be entered in percent of the interval width. Examples:

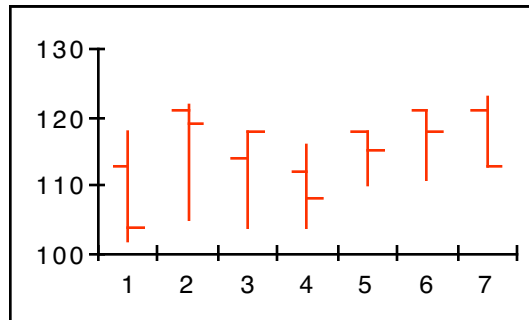
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(26 19 19 24 15 15 6; // high-values
            18 12  8 15 11  6 2) // low-values
  HighLowChart(;on;highLow;;right;;left)
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(118 122 118 116 118 121 123; // high-values
            102 105 104 104 110 111 113; // low-values
            104 119 118 108 115 118 113; // close-values
            113 121 114 112 118 121 121) // open-values
  HighLowChart(;on;highLowCloseOpen;0;;0;;;right;;left)
  GridLocation(all;none) // hide grid
CloseDrawing()

```



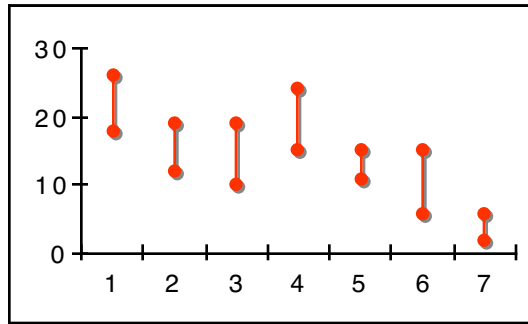
The 1st argument *appearanceConstants* makes it possible to rotate the chart 90 degrees (*appearanceConstants=horizontal*) to add symbols (*appearanceConstants=symbol*), shadow (*appearanceConstants=shadow*) and labels (*appearanceConstants=label*) to the range lines. All options can be combined by using a plus sign "+".

The lines, symbols and shadows can be varied by using the style functions *LineStyle()*, *SymbolStyle()* and *ShadowStyle()* and the labels by using the four style functions *LabelTexts()*, *LabelStyle()*, *LabelBackground()* and *LabelOptions()*. All style functions are discussed in detail in the *Styles* section. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(26 19 19 24 15 15 6; // high-values
            18 12 10 15 11 6 2) // low-values
  HighLowChart(shadow+symbol;on;;0;;0)
  GridLocation(all;none) // hide grid
  SymbolStyle(1;bullet;4)
  ShadowStyle(1;1;gray)
CloseDrawing()

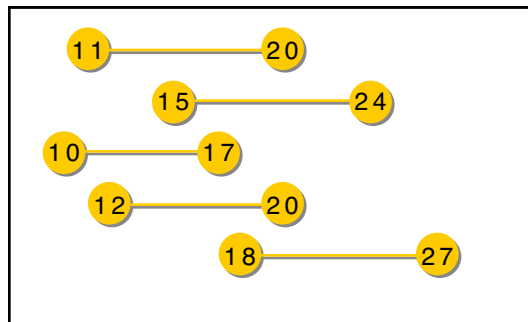
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(27 20 17 24 20; // high-values
            18 12 10 15 11) // low-values
  HighLowChart(horizontal+shadow+label+symbol)
  LineStyle(1;poly;1;darkYellow)
  SymbolStyle(1;bullet;15;;darkYellow)
  ShadowStyle(1;1;gray)
  LabelOptions(1;centerCenter)
  GridLocation(all;none) // hide grid
  AxisOptions(all;none) // hide axes
CloseDrawing()

```

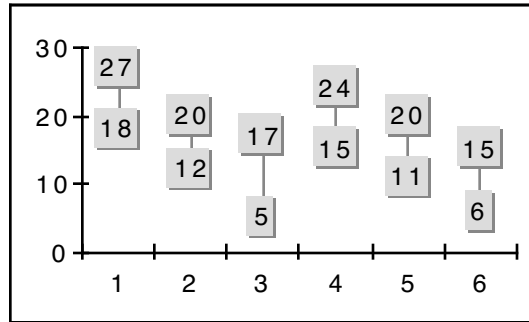


```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(27 20 17 24 20 15; // high-values
            18 12 5 15 11 6) // low-values
  HighLowChart(label;on)
  GridLocation(all;none) // hide grid
  LineStyle(1;poly;1;gray)
  LabelBackground(all;lightGray;;0;;1)
  LabelOptions(all;centerCenter)
CloseDrawing()

```

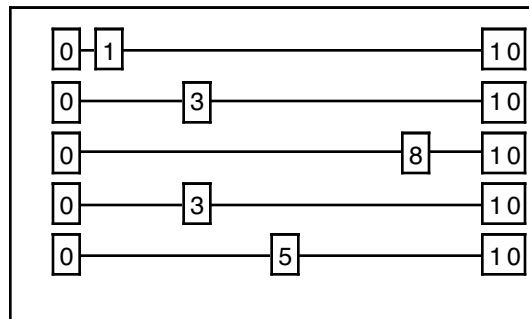




```

OpenDrawing(200;120)
AddFrame(0;0;200;120)
ChartData(10 10 10 10 10; // max-values
          0 0 0 0 0; // min-values
          5 3 8 3 1) // "closing-values"
HighLowChart(label+horizontal;;highLowClose)
LineStyle(1;poly;1;black)
LabelBackground()
LabelOptions(all;centerCenter)
GridLocation(all;none) // hide grid
AxisOptions(all;none) // hide axes
CloseDrawing()

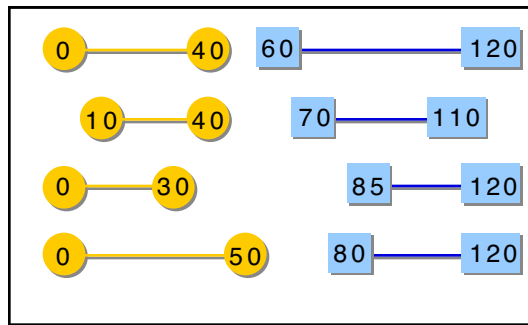
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(50 30 40 40; // 1st series (max-values)
            0 0 10 0; // 1st series (min-values)
            120 120 110 120; // 2nd series (max-values)
            80 85 70 60) // 2nd series (min-values)
  HighLowChart(label+horizontal+symbol+shadow;;;0;;0)
  LineStyle(1;poly;1;darkYellow)
  SymbolStyle(1;bullet;15;;;darkYellow)
  ShadowStyle(all;1;gray)
  SymbolStyle(2;none)
  LabelBackground(2;lightBlue;;;0;;;1)
  LabelOptions(all;centerCenter)
  GridLocation(all;none) // hide grid
  AxisOptions(all;none) // hide axes
CloseDrawing()

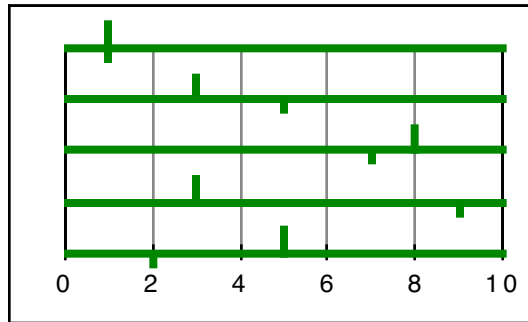
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(10 10 10 10 10; // max-values
            0 0 0 0 0; // min-values
            5 3 8 3 1; // "closing-values"
            2 9 7 5 1) // "opening-values"
  HighLowChart(horizontal;off;highLowCloseOpen;
               0;; // max
               0;; // min
               50;left; // close
               20;right) // open
  AxisOptions(y;none) // hide y-axis
  LineStyle(1;poly;3;green)
  GridFrame()
CloseDrawing()

```



```
HighLowChart2D(appearanceConstants;  
                 highLowChartType;highMarkerLength;  
                 highMarkerAlignment;lowMarkerLength;  
                 lowMarkerAlignment;closeMarkerLength;  
                 closeMarkerAlignment;openMarkerLength;  
                 openMarkerAlignment)
```

The `HighLowChart2D()` function makes it possible to depict value ranges along a time-axis in a variety of ways. At least three data series are necessary. The date/time values are passed to the `ChartData()` function as the 1st data series, the highest values are entered as the 2nd data series, the lowest values as the 3rd data series.

As an option, by using the 2nd argument *highLowChartType*, three values (*highLowChartType=highLowClose*) or even four values (*highLowChartType=highLowCloseOpen*) can be entered in addition to the two default values (*highLowChartType=highLow*).

Example:

```
OpenDrawing(200;120)  
  AddFrame(0;0;200;120)  
  ChartData(2009-11-30 2009-12-01 2009-12-04  
            2009-12-06 2009-12-07 2009-12-08;  
            112 120 110 112 118 120 119;  
            104 105 104 104 110 111 113)  
  HighLowChart2D(;highLow;40;;40)  
  AxisMajorTickLabelStyle(x;;;;;-90)  
  GridLocation(all;none) // hide grid  
CloseDrawing()
```

The `HighLowChart2D()` function is set up the same as the `HighLowChart()` function except the missing argument *doShiftIntervals*.

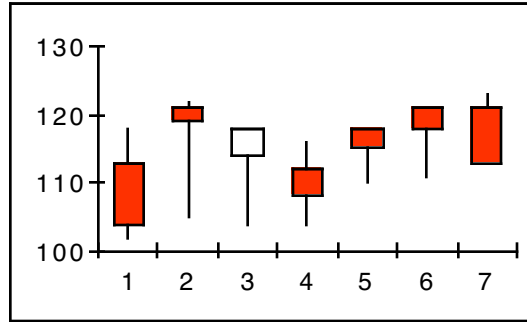
### Candlestick charts:

Candlestick charts can be drawn both as one and two-dimensional.

**`CandlestickChart(appearanceConstants;itemGap;  
highMarkerLength;highMarkerAlignment;  
lowMarkerLength;lowMarkerAlignment)`**

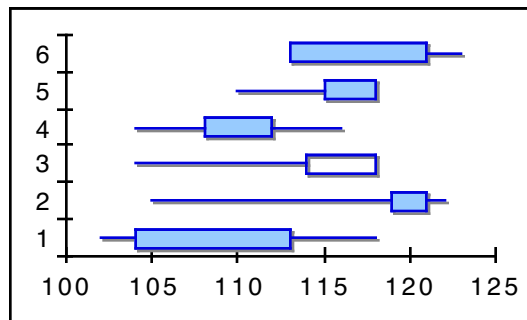
The `CandlestickChart()` function is used to depict stock market prices. The four data series required for this in the `ChartData()` function correspond to the highest, lowest, closing and opening prices. The highest and lowest market price are connected by a line and the range between the opening and closing price by a rectangle. If the closing price is lower than the opening price, the rectangular area is filled, and vice-versa, if the closing price is higher than the opening price, the rectangular area is not filled. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(118 122 118 116 118 121 123; // high-values
            102 105 104 104 110 111 113; // low-values
            104 119 118 108 115 118 113; // close-values
            113 121 114 112 118 121 121) // open-values
  CandlestickChart()
  GridLocation(all;none) // hide grid
CloseDrawing()
```



The 1st argument *appearanceConstants* makes it possible to rotate the chart 90 degrees (*appearanceConstants=horizontal*) and to add shadow (*appearanceConstants=shadow*) to the candlesticks. The appearance constants can be combined by using a plus sign "+". The fill, border and shadow can be varied by using the style functions *FillStyle()*, *PictureStyle()*, *BorderStyle()* and *ShadowStyle()*. All style functions are discussed in detail in the *Styles* section. Examples:

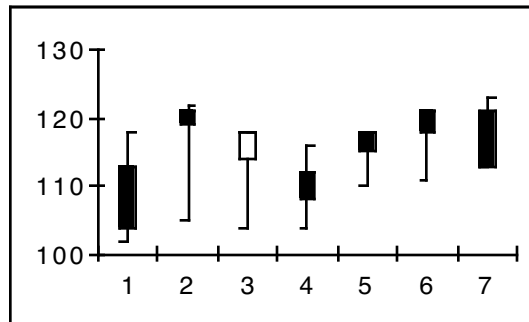
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(118 122 118 116 118 123; // high-values
            102 105 104 104 110 113; // low-values
            104 119 118 108 115 113; // close-values
            113 121 114 112 118 121) // open-values
  CandlestickChart(shadow+horizontal)
  FillStyle(1;lightBlue)
  BorderStyle(1;;1;blue)
  ShadowStyle(1;1;gray)
  GridLocation(all;none) // hide grid
CloseDrawing()
```



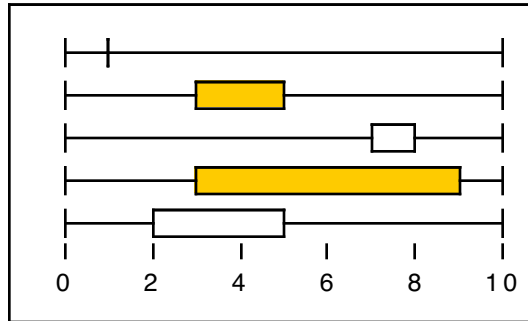
By using the 2nd argument *itemGap*, the distance between the individual candlesticks and the width of the rectangle can be controlled. The space should be entered in percent of the rectangle width. As the default, the space is equal to the rectangle width (*itemGap=100*). For values greater

than 100 (e.g. *itemGap=400*) the rectangles are narrower and the spaces between them larger; for a value less than 100 (e.g. *itemGap=50*) the rectangles are wider and the spaces smaller. As an option markers can be added to the candlesticks. By using the 3rd and 4th argument *high-MarkerLength* and *highMarkerAlignment*, the length and alignment of the markers for the highest values are defined; the 5th and 6th argument *lowMarkerLength* and *lowMarkerAlignment* are used to define the length and alignment of the markers for the lowest values. The length should be entered in percent of the rectangle width. The markers can be aligned to the left, to the right or centered (default). Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(118 122 118 116 118 121 123; // high-values
            102 105 104 104 110 111 113; // low-values
            104 119 118 108 115 118 113; // close-values
            113 121 114 112 118 121 121) // open-values
  CandlestickChart(;300;50;right;50;left)
  FillStyle(1;black)
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(10 10 10 10 10; // high-values
            0 0 0 0 0; // low-values
            5 3 8 3 1; // close-values
            2 9 7 5 1) // open-values
  CandlestickChart(horizontal;60;100;center;100;center)
  FillStyle(1;darkYellow)
  GridLocation(all;none) // hide grid
  AxisLine(x;0) // hide x-axis line
  AxisMajorTicks(x;;;out) // move x-tick marks
  AxisOptions(y;none) // hide y-axis
CloseDrawing()
```



**CandlestickChart2D(**appearanceConstants;itemGap;  
highMarkerLength;highMarkerAlignment;  
lowMarkerLength;lowMarkerAlignment)

The CandlestickChart2D() function is used to depict stock market prices along a time axis. The five data series required for this in the ChartData() function correspond to the date/time values and the high-est, lowest, closing and opening prices. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(2009-11-30 2009-12-01 2009-12-04
            2009-12-06 2009-12-07 2009-12-08;
            118 122 118 116 118 121 123;
            102 105 104 104 110 111 113;
            104 119 118 108 115 118 113;
            113 121 114 112 118 121 121)
  CandlestickChart2D(;300;50;right;50;left)
  FillStyle(1;black)
  AxisMajorTickLabelTexts(x;"|Mon-DD|")
  AxisMajorTickLabelStyle(x;;;;;-90)
  GridLocation(all;none) // hide grid
CloseDrawing()
```

The `CandlestickChart2D()` function is set up the same as the `CandlestickChart()` function.

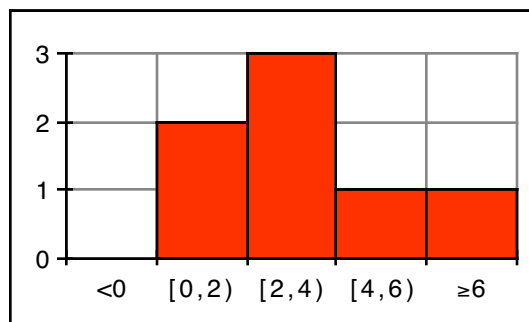
### Histograms:

**Histogram(appearanceConstants;categoryGap;seriesGap)**

**HistogramRange(minValue;maxValue;numOfBins)**

The `Histogram()` function serves to evaluate and depict frequency distributions. As the default, the value range is determined by the largest and smallest value in the `ChartData()` function and divided into ten bins (intervals). By using the arguments *minValue*, *maxValue* and *numOfBins* in the `HistogramRange()` function, both the range to be evaluated and the number of bins can be defined by the user. Please note that, as the default, two end intervals are added, i.e. when, for example, *numOfBins*=3, a total of five bins are depicted. Bins are labeled by the interval limits entered in brackets. For example, `[2,4)` means the bin includes all values greater or equal than two and less than four. The `HistogramRange()` function should be entered after the `Histogram()` function. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(6.5 0 2.9 1.5 3 2.5 5.9)
  Histogram()
  HistogramRange(0;6;3) // after Histogram()!
CloseDrawing()
```

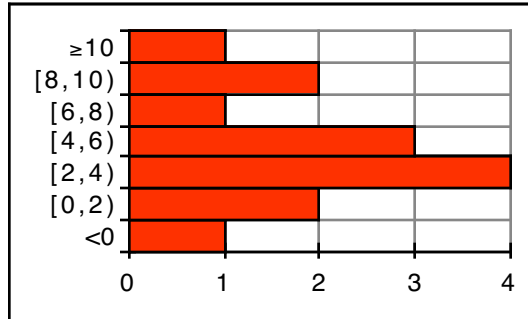




```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 2 0 3 8 4 5 -2 6 9 5 2 12 2)
  Histogram(horizontal)
  HistogramRange(0;10;5) // after Histogram()!
  ScalingOptions(x;;on) // use integers only
CloseDrawing()

```

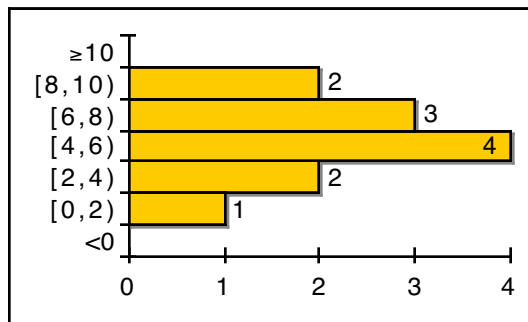


The appearance of the bars is controlled by the arguments *appearance-Constants*, *categoryGap* and *seriesGap* in the `Histogram()` function. These arguments are discussed in detail in combination with the `Bar-Chart()` function. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(2.1 4.8 0 7 9.9 5 7.2 6 4 8 2.5 5.1)
  Histogram(horizontal+shadow+label)
  HistogramRange(0;10;5) // after Histogram()!
  FillStyle(1;darkYellow)
  ShadowStyle(1;1;gray)
  ScalingOptions(x;;on) // integers only
  GridLocation(xy;none) // hide grid
CloseDrawing()

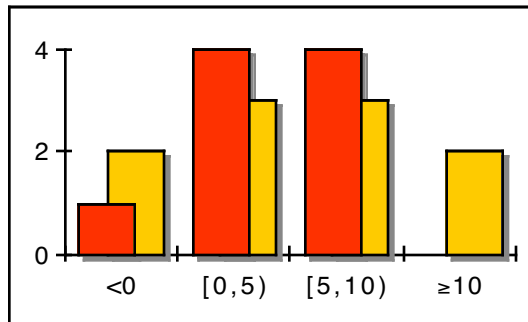
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 0 9.9 5 7.2 3 0.9 6 -.5; // 1st series
            6 9 2 7 2 12 -2 0.5 11 -1) // 2nd series
  Histogram(shadow;50;-50)
  HistogramRange(0;10;2) // after Histogram()!
  FillStyle(2;darkYellow)
  ShadowStyle(all;2;gray)
  ScalingOptions(y;;on) // integers only
  GridLocation(xy;none) // hide grid
CloseDrawing()

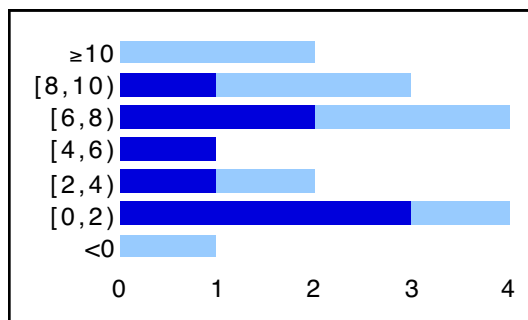
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 0 9.9 5 7.2 3 0.9 6; // 1st series
            6 9 8 7 2 12 -2 0.5 11) // 2nd series
  Histogram(horizontal+stacked;40)
  HistogramRange(0;10;5) // after Histogram()!
  FillStyle(1;blue)
  FillStyle(2;lightBlue)
  BorderStyle(all;none)
  AxisLine(all;0) // hide axis lines
  AxisMajorTicks(all;0) // hide tick marks
  ScalingOptions(x;;on) // integers only
  GridLocation(xy;none) // hide grid
CloseDrawing()

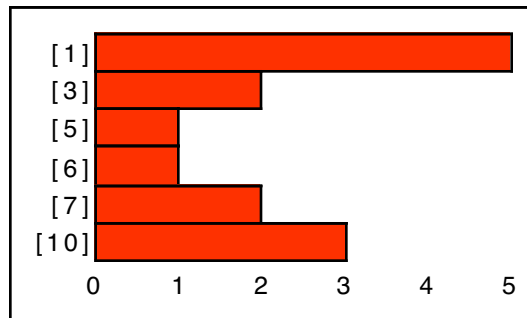
```



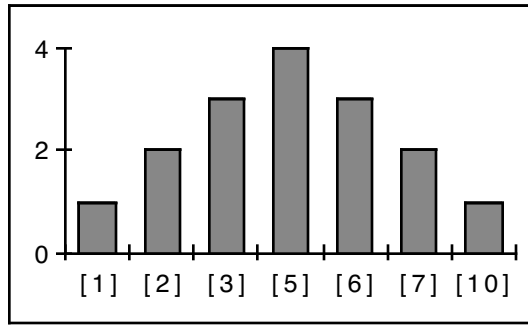
**HistogramOptions(doCountData;doMoveHigher;  
doIncludeEnds;frequencyLineOptions)**

As an option, values can also be evaluated without using interval ranges, but solely according to their frequencies. This is made possible by activating the 1st argument *doCountData=on*. The *HistogramRange()* function is ignored in this case, just as the arguments *doMoveHigher* and *doIncludeEnds*. The *HistogramOptions()* function should be entered after the *Histogram()* function. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 10 7 10 1 5 3 7 3 1 1 6 10 1)
  Histogram(horizontal)
  HistogramOptions(on) // after Histogram()!
  AxisLine(all;0)      // hide axis lines
  AxisMajorTicks(all;0) // hide tick marks
  ScalingOptions(y;on) // y-scaling top to bottom
  ScalingOptions(x;on) // integers only
  GridLocation(xy;none) // hide grid
CloseDrawing()
```

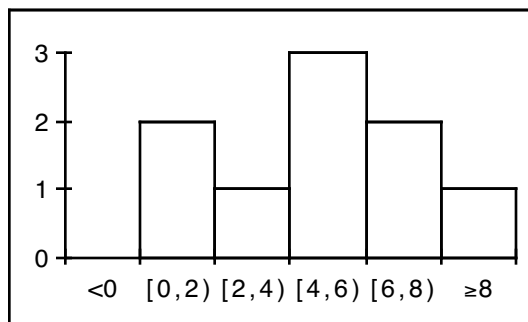


```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(6 2 5 7 3 6 5 3 7 3 2 5 1 6 10 5)
  Histogram(;80)
  HistogramOptions(on) // after Histogram()!
  FillStyle(1;gray)
  GridLocation(xy;none) // hide grid
CloseDrawing()
```



By using the 2nd argument *doMoveHigher*, the assignment of the values, which are located exactly on an interval limit, can be controlled. As the default, these "limit values" are assigned to the higher bin (*doMoveHigher=on*), when *doMoveHigher=off* they are assigned to the lower bin. Examples:

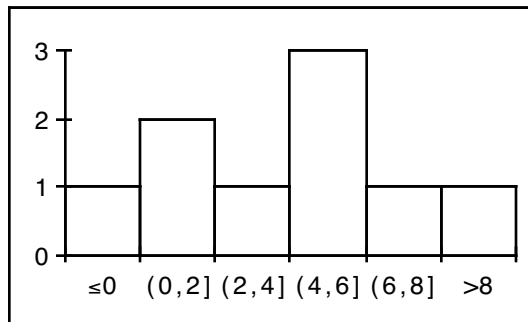
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 0 9 5 7 4 2 6 5)
  Histogram()
  HistogramRange(0;8;4) // after Histogram()!
  HistogramOptions(;on) // after Histogram()!
  FillStyle(1;;transparent)
  ScalingOptions(y;;on) // integers only
  GridLocation(xy;none) // hide grid
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 0 9 5 7 4 2 6 5)
  Histogram()
  HistogramRange(0;8;4) // after Histogram()!
  HistogramOptions(;off) // after Histogram()!
  FillStyle(1;;transparent)
  ScalingOptions(y;;on) // integers only
  GridLocation(xy;none) // hide grid
CloseDrawing()

```

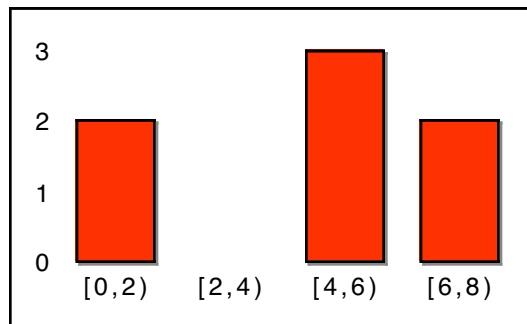


The additional two end bins can be suppressed by using the 3rd argument *doIncludeEnds=off*. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 0 9 5 7 4 8 6 5)
  Histogram(shadow;50)
  HistogramRange(0;8;4) // after Histogram()!
  HistogramOptions(;;off) // after Histogram()!
  ShadowStyle(1;1;gray)
  AxisLine(all;0) // hide axis lines
  AxisMajorTicks(all;0) // hide tick marks
  ScalingOptions(x;;on) // integers only
  GridLocation(xy;none) // hide grid
CloseDrawing()

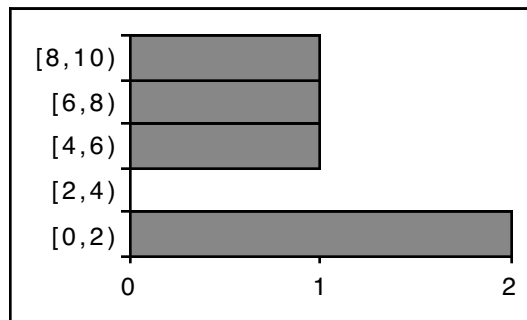
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 0 9.9 5 7.2)
  Histogram(horizontal)
  HistogramOptions(;;off) // after Histogram()!
  HistogramRange(0;10;5) // after Histogram()!
  FillStyle(all;gray)
  GridLocation(xy;none)
  ScalingOptions(x;;on) // integers only
CloseDrawing()

```



As an option, frequency lines can be added to histograms. As the 4th argument *frequencyLineOptions*, the following constants are available for this:

<i>constant</i>	<i>value</i>	<i>note</i>
none	0	no frequency lines
frequency	1	frequency lines
ogive	2	running totaled frequencies
reverseOgive	3	backwards running totaled frequencies

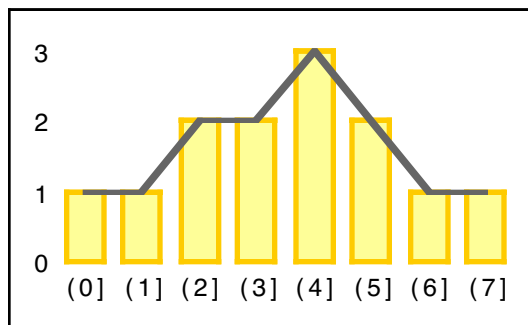
The appearance of the frequency lines can be controlled by the `LineStyle()` function. The `LineStyle()` function is explained in the *Styles* section.

## Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 4 9.9 5 4.2 3 4.9 6
            6 9 8 7 2 12 3 3.5 11)
  Histogram(;50)
  HistogramRange(0;8;8) // after Histogram()!
  HistogramOptions(;off;off;frequency)
  FillStyle(1;lightYellow)
  BorderStyle(1;;2;darkYellow)
  LineStyle(1;poly;2;darkGray)
  GridLocation(xy;none) // hide grid
  AxisLine(all;0)       // hide axis lines
  AxisMajorTicks(all;0) // hide tick marks
  AxisMajorTickLabelTexts(x;"(|u|]")
CloseDrawing()

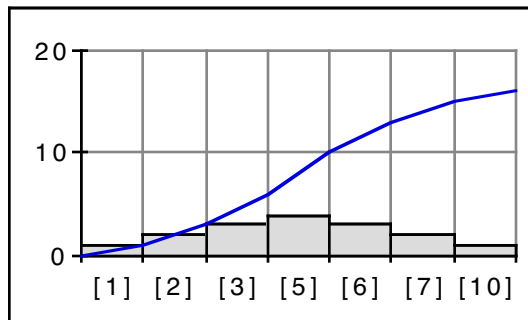
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(6 2 5 7 3 6 5 3 7 3 2 5 1 6 10 5)
  Histogram()
  HistogramOptions(on;;;ogive) // after Histogram()!
  FillStyle(1;lightGray)
  LineStyle(1;poly;1;blue)
CloseDrawing()

```



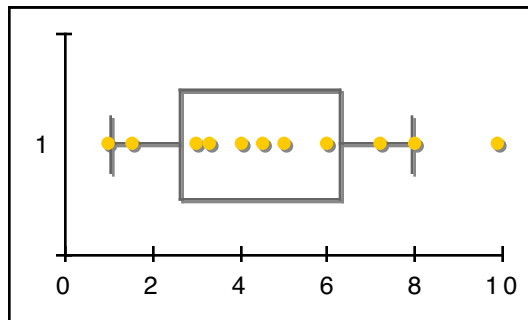
**Box Plots:**

Box plots, also referred to as box and whisker plots, are used to represent statistical distributions.

**BoxPlot(*appearanceConstants*;upperBoxPercentile;  
lowerBoxPercentile;upperWhiskerPercentile;  
lowerWhiskerPercentile)**

The 1st argument *appearanceOptions* makes it possible to rotate the chart 90 degrees (*appearanceOptions=horizontal*), to add symbols (*appearanceOptions=symbol*) and shadow (*appearanceOptions=shadow*) to the plots. All options can be combined by using a plus sign "+". The borders, symbols and shadows can be varied by using the style functions *BorderStyle()*, *SymbolStyle()* and *ShadowStyle()*. All style functions are discussed in detail in the *Styles* section. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 1.5 8 9.9 4.5 7.2 3.3 4 5 3 6)
  BoxPlot(symbol+shadow+horizontal)
  SymbolStyle(all;bullet;4;1;darkYellow)
  BorderStyle(all;;1;darkGray)
  ShadowStyle(all;1;gray)
  GridLocation(xy;none) // hide grid
CloseDrawing()
```

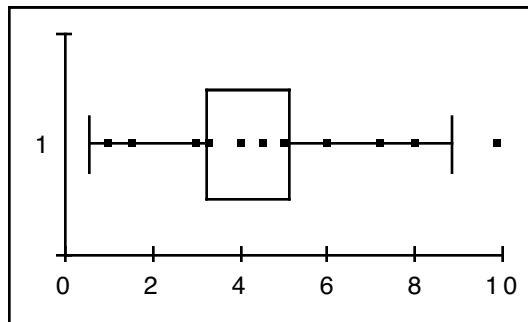


The box limits are defined by the arguments *upperBoxPercentile* (default: 75 percent) and *lowerBoxPercentile* (default: 25 percent), the whisker limits are defined by the arguments *upperWhiskerPercentile* (default: 90 percent) and *lowerWhiskerPercentile* (default: 10 percent).



Example:

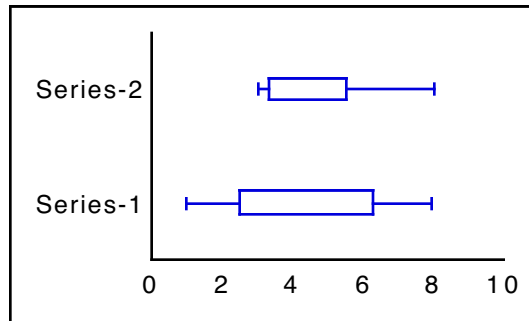
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 1.5 8 9.9 4.5 7.2 3.3 4 5 3 6)
  BoxPlot(symbol+horizontal;65;35;95;5)
  SymbolStyle(all;square;2;1;black)
  GridLocation(xy;none) // hide grid
CloseDrawing()
```



**BoxPlotOptions(itemGap;isPercentileGraph;doFillBox;  
showMean;showMedian;showOutliersOnly;  
showCapsOnly;capLength;confidenceInterval)**

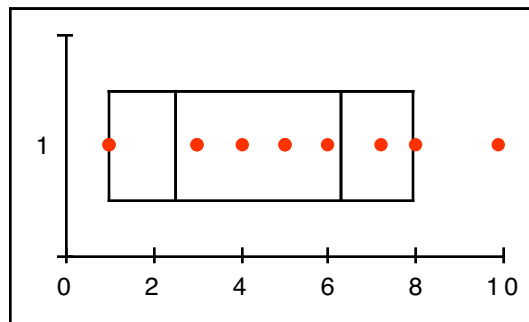
The BoxPlotOptions() function should be listed after the BoxPlot() function and opens up a variety of options. By using the 1st argument *itemGap*, the space between the individual series can be controlled. The space is entered in percent of the box width. As the default, the space equals the box width (*itemGap=100*). For values greater than 100 (e.g. *itemGap=400*) the boxes are narrower and the spaces between them bigger; for values less than 100 (e.g. *itemGap=50*) the boxes are wider and the spaces smaller. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 1 8 9.9 5 7.2 3 4 5 3 6; // 1st series
            3 5 3.2 4.1 8 9 5 3.5 4.9 6.1) // 2nd series
  BoxPlot(horizontal)
  BoxPlotOptions(400) // after BoxPlot(!)
  BorderStyle(all;;1;blue)
  AxisMajorTicks(all;0)
  AxisMajorTickLabelTexts(y;"Series-1";"Series-2")
  GridLocation(xy;none) // hide grid
CloseDrawing()
```



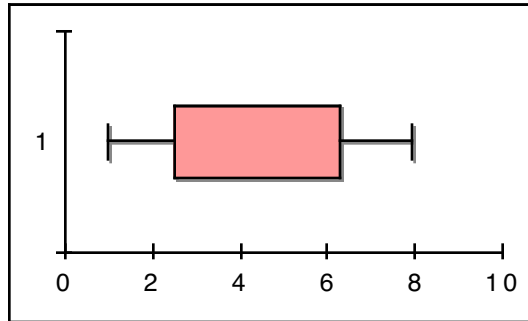
By activating the 2nd argument *isPercentileGraph=on*, the box is extended to the whisker percentile. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 1 8 9.9 5 7.2 3 4 5 3 6)
  BoxPlot(horizontal+symbol)
  BoxPlotOptions(;on) // after BoxPlot()!
  SymbolStyle(1;bullet;4)
  GridLocation(xy;none) // hide grid
CloseDrawing()
```



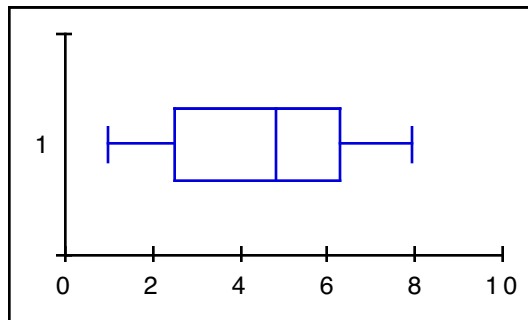
The fill of the box can be controlled by using the 3rd argument *doFillBox=on* and the *FillStyle()* or *PictureStyle()* function. The *FillStyle()* and *PictureStyle()* functions are explained in the *Styles* section. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 1 8 9.9 5 7.2 3 4 5 3 6)
  BoxPlot(horizontal+shadow)
  BoxPlotOptions(200;off;on) // after BoxPlot()!
  FillStyle(1;lightRed)
  BorderStyle(1;;1)
  ShadowStyle(1;1;gray)
  GridLocation(xy;none) // hide grid
CloseDrawing()
```

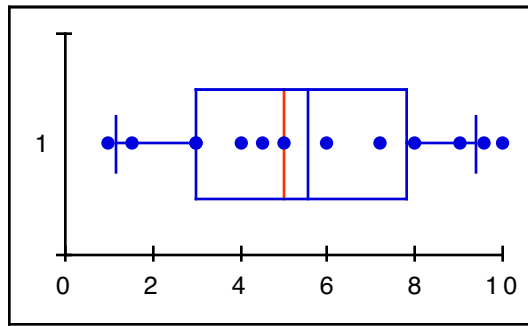


Furthermore, marker lines can be added for the arithmetic mean and median value by activating the arguments *showMean=on* and *showMedian=on*. The depiction of the mean value line is identical to the border and is controlled by the `BorderStyle()` function. The appearance of the median line can be controlled by the `LineStyle()` function. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 1 8 9.9 5 7.2 3 4 5 3 6)
  BoxPlot(horizontal)
  BoxPlotOptions(200;off;off;on) // after BoxPlot()!
  BorderStyle(1;;1;blue)
  GridLocation(xy;none) // hide grid
CloseDrawing()
```

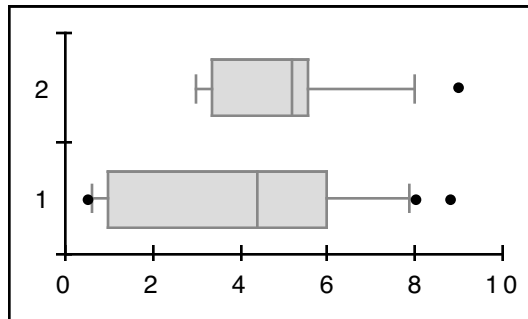


```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 1 8 9.6 5 7.2 10 9 3 4 4.5 3 6)
  BoxPlot(horizontal+symbol)
  BoxPlotOptions(100;off;off;on;on) // after BoxPlot()!
  BorderStyle(1;;1;blue)
  LineStyle(1;;1;red)
  SymbolStyle(1;bullet;4;1;blue)
  GridLocation(xy;none)
CloseDrawing()
```



By activating the argument *showOutliersOnly=on*, the depiction of the symbols is limited to the "outliers", i.e. to those values which lie outside of the whisker percentile. The depiction of symbols is activated by *appearanceOptions=symbol* in the `BoxPlot()` function. Example:

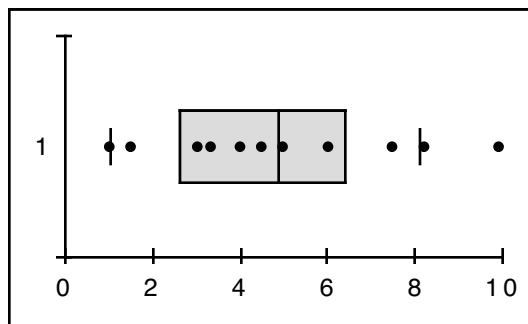
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(0.5 1 1 8 8.8 5 7.2 3 4 5 3 6; // 1st series
            3 5 3.2 4.1 8 9 5 3.5 4.9 6.1) // 2nd series
  BoxPlot(horizontal+symbol)
  BoxPlotOptions(100;off;on;on;off;on) //after BoxPlot()
  FillStyle(all;lightGray)
  BorderStyle(all;;1;gray)
  SymbolStyle(all;bullet;3;;black)
  GridLocation(xy;none) // hide grid
CloseDrawing()
```



By using the argument *showCapsOnly=on*, the lines between the box and the whisker caps can be suppressed.

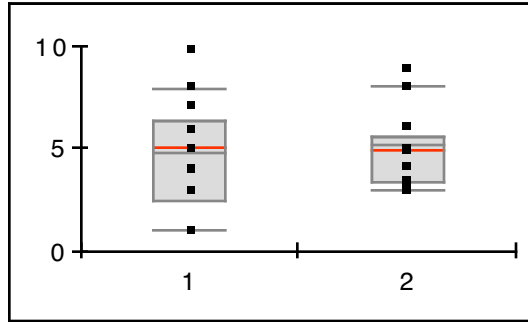
Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 1.5 8.2 9.9 5 7.5 3 4 4.5 3.3 6)
  BoxPlot(horizontal+symbol)
  BoxPlotOptions(200;off;on;on;off;off;on)
  FillStyle(1;lightGray)
  BorderStyle(1;;1;)
  SymbolStyle(1;bullet;3;;black)
  GridLocation(xy;none) // hide grid
CloseDrawing()
```



The marker length of the whisker percentile can be controlled by using the last argument *capLength*. The marker length should be entered in percent of the box width. The default is 50, i.e. half of the width of the box. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 1 8 9.9 5 7.2 3 4 5 3 6;      // 1st series
            3 5 3.2 4.1 8 9 5 3.5 4.9 6.1) // 2nd series
  BoxPlot(symbol)
  BoxPlotOptions(200;off;on;on;on;off;on;100)
  FillStyle(all;lightGray)
  BorderStyle(all;;1;gray)
  LineStyle(all;;1;red) // median
  SymbolStyle(all;square;2;;black)
  GridLocation(xy;none)
CloseDrawing()
```

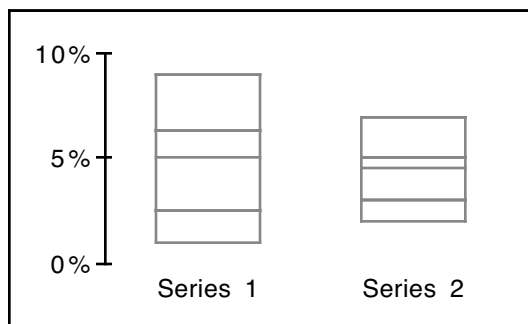


Box plots can also be used to represent quartiles. The upper box percentile is set to equal 75% (that corresponds to the 3rd quartile) and the lower box percentile to equal 25% (that corresponds to the 1st quartile). The 2nd quartile (=50% percentile) corresponds to the median. The quartiles are limited by the upper whisker percentile equaling 100% and by the lower whisker percentile equaling 0%. Example:

```

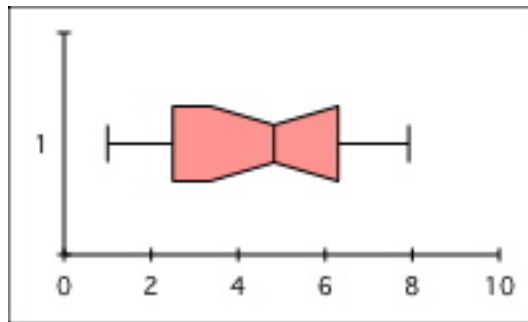
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 1 8 9 5 7.2 3 4 5 3 6;          // 1st Series
            3 5 3 4.1 7 3 5 2 5 3.5 4.9 6) // 2nd Series
  BoxPlot(;75;25;100;0)
  BoxPlotOptions(;on; // isPercentileGraph
                 off; // no fill
                 off; // no mean
                 on) // median (= 2nd quartile)
  BorderStyle(all;;1;gray)
  LineStyle(all;;1;gray) // median
  AxisLine(x;0)
  AxisMajorTicks(x;0)
  AxisMajorTickLabelTexts(y;"|u|%" )
  GridLocation(xy;none)
  AxisMajorTickLabelTexts(x;"Series |u|")
CloseDrawing()

```



As an option, a predefined confidence interval can be made visible by notching the rectangular box using the argument *confidenceInterval*.

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 1 8 9.9 5 7.2 3 4 5 3 6)
  BoxPlot(horizontal)
  BoxPlotOptions(200;off;on;
                 on; // show mean
                 off;
                 off;
                 off;
                 50; // cap length
                 90) // 90% confidence interval
  FillStyle(1;lightRed)
  GridLocation(xy;none)
CloseDrawing()
```



## Styles

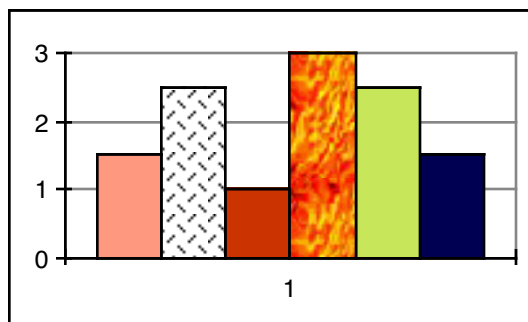
Styles make it possible to control the appearance of each data series. The following components can be designed individually:

- fills
- borders
- lines
- symbols
- shadows
- arrows
- labels

### **FillStyle(seriesIndex;color;pattern)**

By using the `FillStyle()` function, each data series can be assigned a separate fill color and a fill pattern. Series, which are not explicitly assigned a fill style, are represented in one of the 16 default fill colors. An overview of the predefined default colors can be found in *xmReference*. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5; 2.5; 1; 3; 2.5; 1.5)
  BarChart()
  FillStyle(1;red;gray)
  FillStyle(2;black;48)
  FillStyle(3;darkRed)
  FillStyle(4;;127)
  FillStyle(5;200 231 89)
CloseDrawing()
```

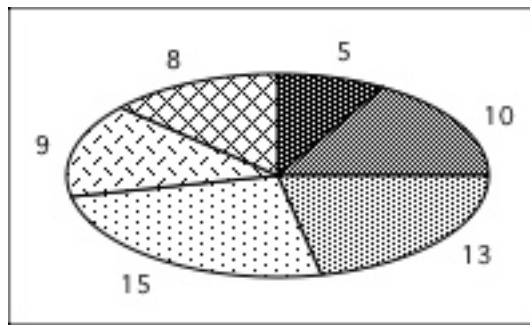




```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 10 13 15 9 8)
  PieChart(oval+label)
  BorderStyle(all;)
  FillStyle(1;black;darkGray)
  FillStyle(2;black;gray)
  FillStyle(3;black;lightGray)
  FillStyle(4;black;13)
  FillStyle(5;black;48)
  FillStyle(6;black;53)
CloseDrawing()

```

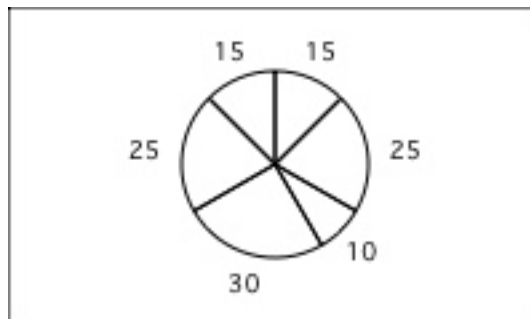


If no series index is defined or if *seriesIndex=all* is set, all series are assigned the same fill style. For example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(15 25 10 30 25 15)
  PieChart(label)
  FillStyle(;;transparent)
CloseDrawing()

```



**PictureStyle(seriesIndex;sourceType;sourceName;  
stackAndScaleAt)**

Instead of fill colors, pictures can also be used as fills. Three different pictures sources are supported:

- *Clipboard:*

The clipboard is used when the constant *clipboard* is entered as the *sourceType*. The argument *sourceName* is ignored.

Example: `PictureStyle(1;clipboard)`

- *File:*

Either a complete, absolute file path or only a relative path can be passed. The relative path refers to the folder in which the current FileMaker Pro database file is located. Examples:

`PictureStyle(1;file;"Pictures/Pict_01.png")`

`PictureStyle(2;file;"C:/Pictures/Pict_01.png")`

`PictureStyle(3;file;"Macintosh HD/Picts/Pict_01.pdf")`

In Mac OS X pictures have to be in PDF, PICT, GIF, JPEG, PNG, BMP, TIFF format in order to be imported.

In Windows pictures have to be in WMF, EMF, GIF, JPEG, PNG, BMP, TIFF format in order to be imported.

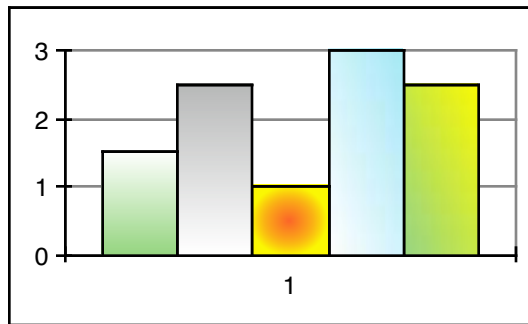
- *Resource:*

Presently, a total of 42 gradient fills stored in resources are available. Access is gained by using an index number between 1 and 42. Please note that the index number must be placed in double quotes. An overview of the predefined gradient fill resources can be found in *xmReference*.

Furthermore, when using resources in Windows, please note that charts can grow to a size consisting of several MB. This problem only arises in connection with the standard output format EMF and not with the bitmap output format. EMF and bitmap format are explained along with the `OpenDrawing()` function in the *Layout* section.

Examples:

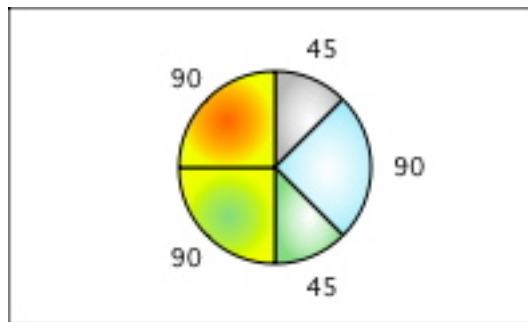
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5; 2.5; 1; 3; 2.5)
  BarChart()
  PictureStyle(1;resource;"5")
  PictureStyle(2;resource;"2")
  PictureStyle(3;resource;"42")
  PictureStyle(4;resource;"24")
  PictureStyle(5;resource;"30")
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(45 90 45 90 90)
  PieChart(label)
  PictureStyle(1;resource;"34")
  PictureStyle(2;resource;"36")
  PictureStyle(3;resource;"38")
  PictureStyle(4;resource;"40")
  PictureStyle(5;resource;"42")
CloseDrawing()

```

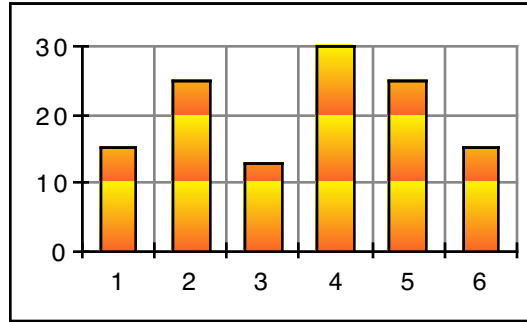


For bar charts there is also the possibility to scale and stack pictures by defining a scaling value via the argument *stackAndScaleAt*. Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(15 25 13 30 25 15)
  BarChart()
  PictureStyle(all;resource;"8";10)
CloseDrawing()

```



### **BorderStyle(seriesIndex;appearance;width;color;pattern)**

As the default, all borders are black and one-pixel wide. By using the `BorderStyle()` function, each data series can be assigned a separate border. In addition, the border of some charts can be varied using the argument *shape*.

<i>shape</i>	<i>value</i>	<i>chart type</i>
none	0	all charts
jump	1	(not supported)
step	2	area chart
poly	3	all charts (default)
smooth	4	area chart, radar chart, polar chart

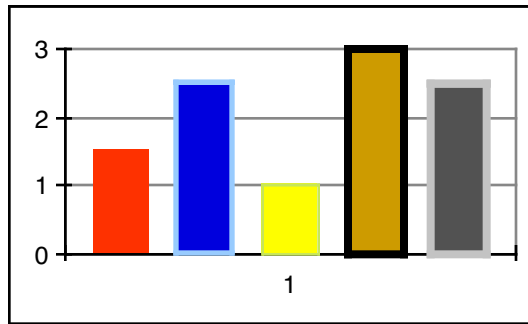
The degree of smoothing (*shape*=4) can be controlled by an additional parameter. The value range lies between 0.0 (no smoothing, polygonal) and 2.0 (strong smoothing); the default value is 1. Furthermore, a dash pattern can be assigned to the border width by adding a list of dash lengths and gaps. Examples can be found under `LineStyle()`.

Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5; 2.5; 1; 3; 2.5)
  BarChart(;;50)
  BorderStyle(1;poly;0)
  BorderStyle(2;poly;2;lightBlue)
  BorderStyle(3;poly;1;200 231 89)
  BorderStyle(4;;3)
  BorderStyle(5;;3;gray;gray)
CloseDrawing()

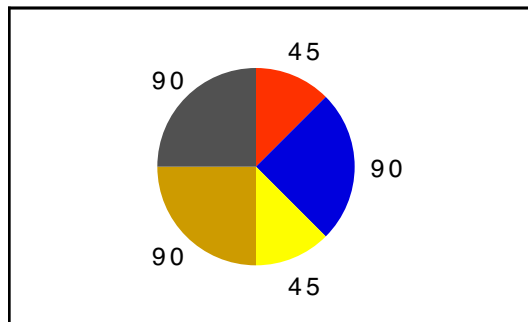
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(45 90 45 90 90)
  PieChart(label)
  BorderStyle(all;none)
CloseDrawing()

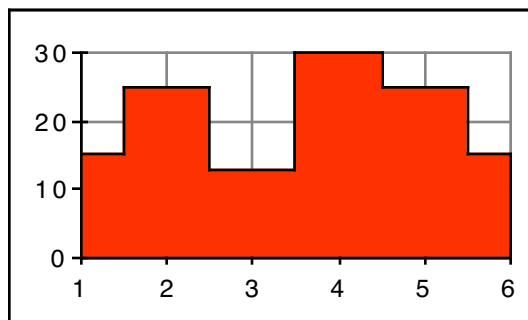
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(15 25 13 30 25 15)
  AreaChart()
  BorderStyle(1;step;1;black)
CloseDrawing()

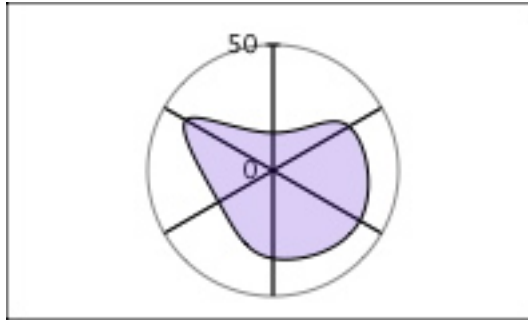
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(15 35 40 35 25 40)
  RadarChart()
  MajorGridLinePatterns(all;all;black)
  FillStyle(1;50 0 200 50)
  BorderStyle(1;smooth)
CloseDrawing()

```



### **LineStyle(seriesIndex;shape;width;color;pattern)**

By using the `LineStyle()` function, each data series can be assigned a separate line style. Series, which are not explicitly assigned a style, are represented in one of the 16 default colors. An overview of the predefined colors can be found in *xmReference*. In addition, the curves can be varied using the argument *shape*.

<i>shape</i>	<i>value</i>	<i>note</i>
none	0	hide lines
jump	1	only horizontal or vertical lines
step	2	step-like lines
poly	3	polygonal lines ( <i>default</i> )
smooth	4	smooth lines

The degree of smoothing (*shape=4*) can be controlled by an additional parameter. The value range lies between 0.0 (no smoothing, polygonal) and 2.0 (strong smoothing); the default value is 1. Examples:

```

LineStyle(all;4;1;darkBlue)      // smoothing: 1.0 (default)
LineStyle(all;4 0;1;darkBlue)    // smoothing: 0.0 (polygonal)
LineStyle(all;4 0.5;1;darkBlue)  // smoothing: 0.5
LineStyle(all;smooth 0.5)        // Error, not allowed!
LineStyle(all;smooth,0.5)        // Error, not allowed!

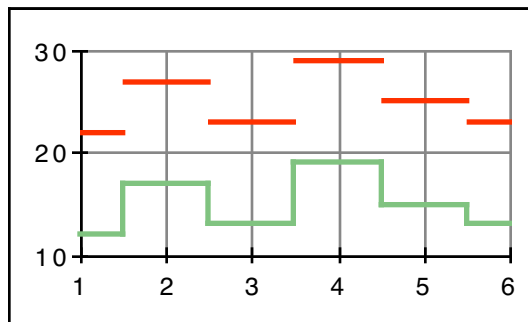
```

Furthermore, a dash pattern can be assigned to the line width by adding a list of dash lengths and gaps.

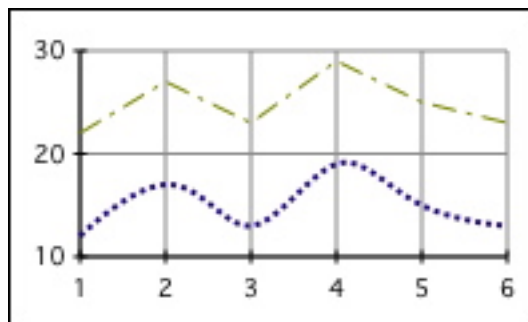
Examples:

```
LineStyle(all;poly;1)           // solid line (default)
LineStyle(all;poly;1 2 2)       // dotted line
LineStyle(all;poly;1 5 5)       // dashed line (5 pixels long)
LineStyle(all;poly;2 9 4 2 4)   // dash-dotted line
```

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(22 27 23 29 25 23;
            12 17 13 19 15 13)
  LineChart()
  LineStyle(1;jump;2)
  LineStyle(2;step;2;green;gray)
CloseDrawing()
```



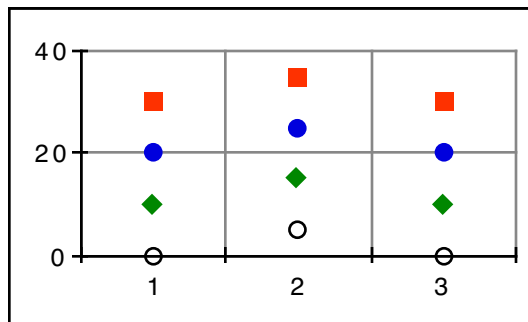
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(22 27 23 29 25 23;
            12 17 13 19 15 13)
  LineChart()
  LineStyle(1;poly;1 9 4 2 4;olive)
  LineStyle(2;smooth;2 2 2;darkBlue)
CloseDrawing()
```



### **SymbolStyle(seriesIndex;type;size;lineWidth;color;pattern)**

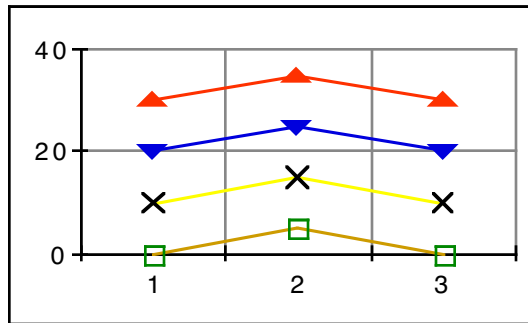
By using the `SymbolStyle()` function, each data series can be assigned a separate symbol. Series, which are not explicitly assigned a symbol, are represented in one of the 12 default symbols. Presently, a total of 18 symbols are available. An overview of the predefined symbols can be found in *xmReference*. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(30 35 30; 20 25 20; 10 15 10; 0 5 0)
  ScatterChart(;on)
  SymbolStyle(1;square;6)
  SymbolStyle(2;bullet;6)
  SymbolStyle(3;diamond;6;;green)
  SymbolStyle(4;circle;6;;black)
CloseDrawing()
```



```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(30 35 30; // 1st series
            20 25 20; // 2nd series
            10 15 10; // 3rd series
            0 5 0) // 4th series
  LineChart(symbol;on)
  SymbolStyle(1;upTriangle;10)
  SymbolStyle(2;downTriangle;10)
  SymbolStyle(3;cross;8;1;black)
  SymbolStyle(4;hollowSquare;6;1;green)
CloseDrawing()
```



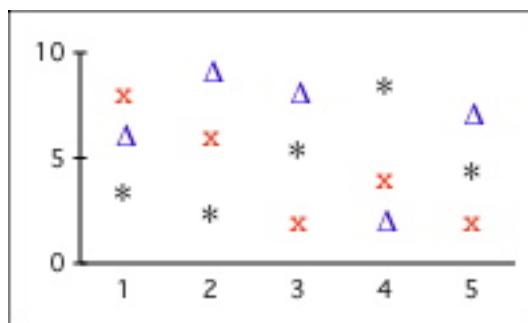


As an option, characters from a character set can also be used as symbols; this results in a myriad of new "symbols" which, when using Postscript or TrueType fonts, can be output with high quality. The characters are defined by the `LabelTexts()` and `LabelStyle()` functions. The `LabelOptions()` function makes it possible to position the characters precisely.

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 2 5 8 4; 8 6 2 4 2; 6 9 8 2 7)
  ScatterChart(label;on)
  SymbolStyle(all;none)
  LabelOptions(all;centerCenter)
  LabelTexts(1;"*")
  LabelStyle(1;"Times";16)
  LabelTexts(2;"x")
  LabelStyle(2;"Courier";12;;red)
  LabelTexts(3;"\u2206") // \xUUUU Unicode (hexadecimal)
  LabelStyle(3;"Times";12;;blue)
  AxisMajorTicks(x;0)
  GridLocation(;none)
CloseDrawing()

```



Symbols can also be displayed when using a bubble chart. By using the `PictureStyle()` function it is possible to read in pictures from a file and use them as symbols. Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 1 6 8; 1 1 1 1;
            8 4 2 4; 1 1 1 1;
            6 9 8 2; 1 1 1 1)
  BubbleChart(;on)
  BubbleChartOptions(9)
  BorderStyle(all;0)
  AxisMajorTicks(x;0)
  GridLocation(;none)

  // Example for Mac OS X:
  // Supported file formats:
  // PDF, PICT, GIF, JPEG, PNG, BMP, TIFF
  PictureStyle(1;file;"Symbols/Symbol01.pdf")
  PictureStyle(2;file;"Macintosh HD/Symbols/Symb1.pdf")

  // Example for Windows:
  // Supported file formats:
  // WMF, EMF, GIF, JPEG, PNG, BMP, TIFF
  PictureStyle(3;file;"C:/Symbols/Symbol03.png")
CloseDrawing()

```

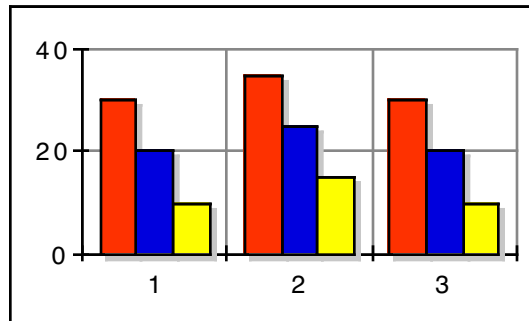
### **ShadowStyle(seriesIndex;offset;color;pattern)**

The default shadow for charts is gray with an offset of three pixels. The shadows can be varied by using the `ShadowStyle()` function. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(30 35 30; // 1st series
            20 25 20; // 2nd series
            10 15 10) // 3rd series
  BarChart(shadow)
  ShadowStyle(all;2;200 200 200)
CloseDrawing()

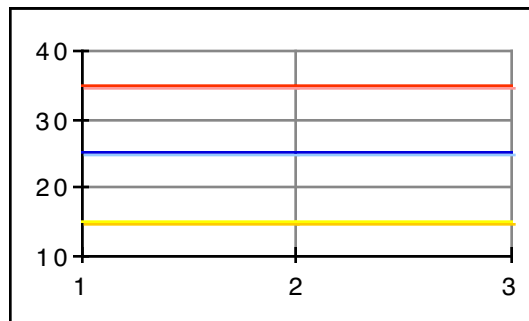
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(35 35 35; 25 25 25;15 15 15)
  LineChart(shadow)
  ShadowStyle(1;1;lightRed)
  ShadowStyle(2;1;lightBlue)
  ShadowStyle(3;1;darkYellow)
CloseDrawing()

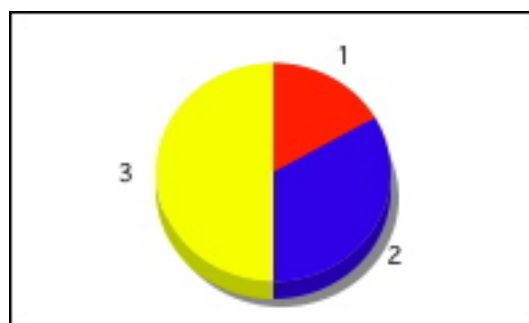
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 2 3)
  PieChart(shadow+label;15)
  BorderStyle(all;none)
  ShadowStyle(all;3;gray)
CloseDrawing()

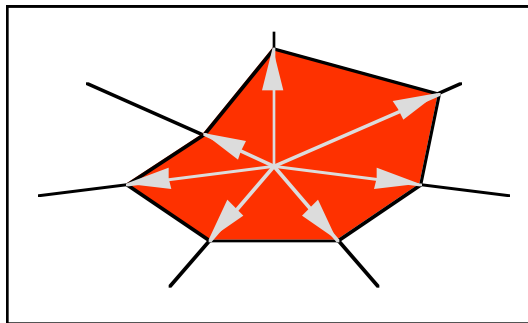
```



**ArrowStyle(seriesIndex;lineWidth;lineColor;linePattern;  
headLocation;headLength;headWidth;headInset;  
hasHollowHead)**

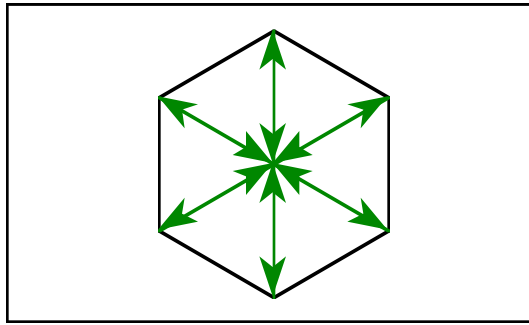
As an option arrows can be added to radar and polar charts. They run from the center to the individual chart points. As the default, all arrows are black and one-pixel wide. By using the `ArrowStyle()` function, the appearance of the arrows, including the arrowheads, can be controlled separately for each data series. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(35 35 25 25 25 25 15)
  RadarChart(oval)
  // hide scaling and grid
  RadarChartOptions(0;none;on)
  ArrowStyle(1;1;lightGray)
CloseDrawing()
```



Furthermore, the arrowhead can be placed on the end (default), at the beginning or on both sides using the argument *headLocation*. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(40 40 40 40 40 40)
  RadarChart()
  // hide scaling and grid
  RadarChartOptions(0;none;on)
  AxisLine(all;0) // hide axes
  FillStyle(1;;transparent) // no fill
  ArrowStyle(1;1;green;;begin+end;15;10;4)
CloseDrawing()
```

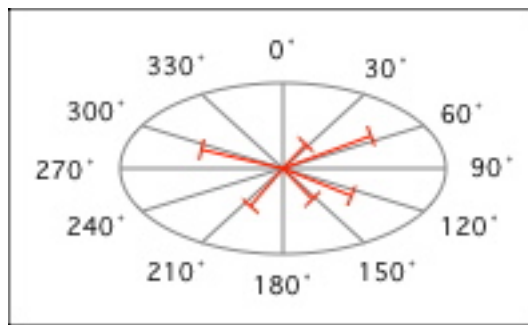


```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(-10 21 7 -25 9 27; -22 -16 14 11 -18 19)
  PolarChart(oval)
  PolarChartOptions(0;;on) // hide scaling
  ArrowStyle(1;2;red;;end;0)
  AxisLabelText(all;"|u|°")

  AxisLine(all;0) // hide axes
  BorderStyle(1;none) // hide border
  FillStyle(1;;transparent) // no fill
CloseDrawing()

```

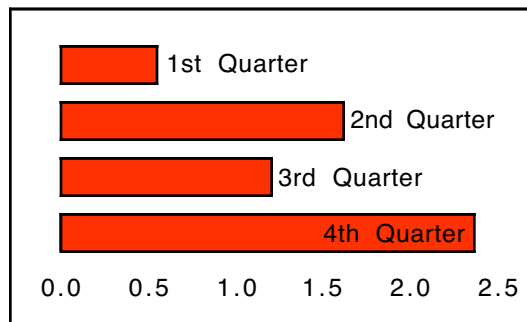


**LabelTexts(seriesIndex;text1;text2...)**

The `LabelTexts()` function makes it possible to assign different texts to the data points of a chart. In doing so, texts can be set up in a variety of ways:

- Text only: For example:

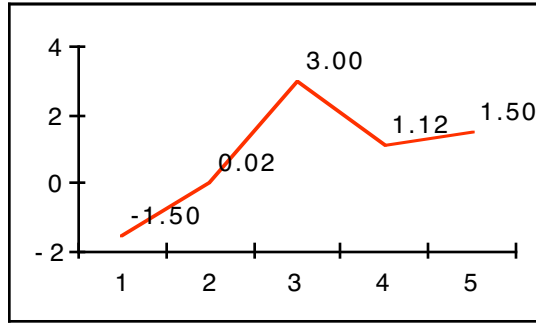
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(0.5555 1.61 1.2 2.365)
  BarChart(label+horizontal;50)
  LabelTexts(1;"1st Quarter";
             "2nd Quarter";
             "3rd Quarter";
             "4th Quarter")
  GridLocation(all;none)
  ScalingOptions(y;on)
  AxisOptions(y;none)
  AxisLine(x;0)
  AxisMajorTicks(all;0)
CloseDrawing()
```



- Format specifier:

A format specifier must always be placed between two vertical bars (pipe characters) "|". A detailed explanation of all format specifiers, including numerous examples, can be found in *xmReference*. Example:

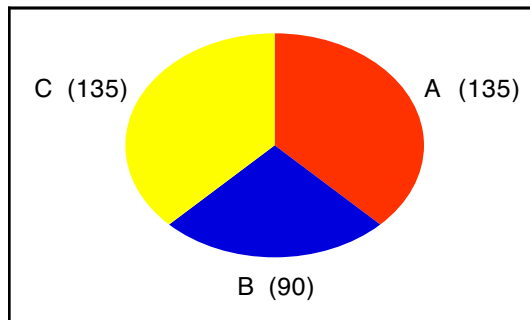
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(-1.5 0.024 3 1.121 1.5)
  LineChart(label;on)
  LabelTexts(1;"|f2|")
  GridLocation(all;none)
CloseDrawing()
```



- Combination of text and format specifier:

As an option, texts can be added before and after a format specifier. Example:

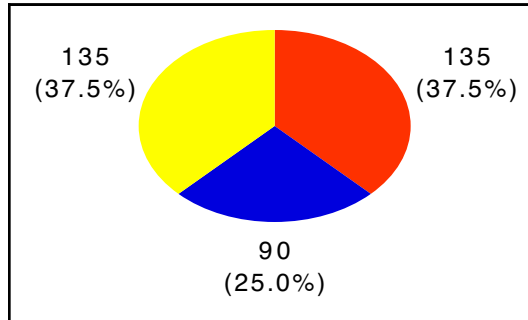
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(135 90 135)
  PieChart(label+oval)
  BorderStyle(all;none)
  LabelTexts(1;"A (|u|)";"B (|u|)";"C (|u|)")
CloseDrawing()
```



On pie charts and stacked charts such as bar charts, area charts and histograms it is also possible to represent both absolute values (default) and percentage values. This can be done by specifying two format specifiers. The first specifier defines the format of the absolute values; the second format specifier serves to format the percentage values.

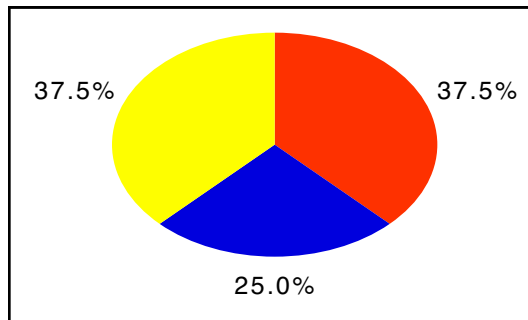
Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(135 90 135)
  PieChart(label+oval)
  BorderStyle(all;none)
  LabelTexts(1;"|u|\n(|f1|%)")
  LabelStyle(all;;;;center)
CloseDrawing()
```



If only the percentage values are to be represented, the output of the absolute values has to be suppressed by an empty format specifier "|".  
Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(135 90 135)
  PieChart(label+oval)
  BorderStyle(all;none)
  LabelTexts(1;"|||f1|%")
CloseDrawing()
```



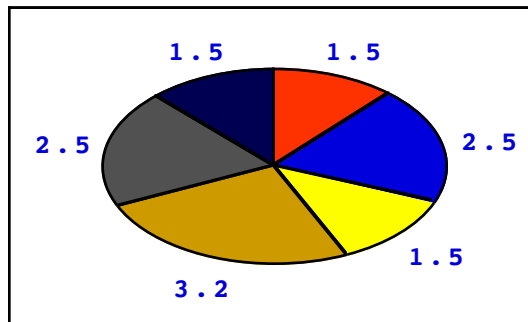
Texts and format specifiers are repeated periodically if the number of chart values is larger than the number of text arguments. If no specifier is entered, the default format specifier "|u|" is used. Furthermore, texts consisting of several lines are possible by inserting a line feed "\n".



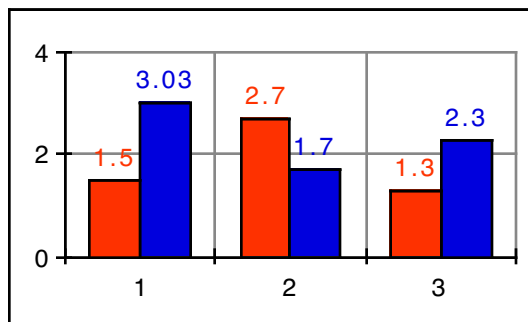
**LabelStyle(seriesIndex;font;size;style;color;alignment;orientation;maxWidth;maxHeight;ellipsisPosition)**

By using the LabelStyle() function, labels can be assigned a text style. In doing so, a separate text style can be defined for each data series. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1.5 3.2 2.5 1.5)
  PieChart(oval+label;)
  LabelStyle(1;"Courier";10;bold;blue)
CloseDrawing()
```

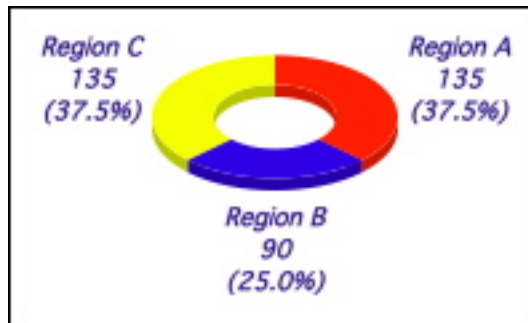


```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.7 1.3; 3.03 1.7 2.3)
  BarChart(label)
  LabelStyle(1;;9;;red)
  LabelStyle(2;;9;;blue)
CloseDrawing()
```



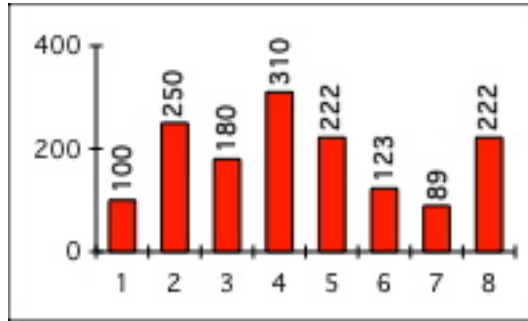
By using the argument *alignment* multi-line texts can either be aligned to the left (*alignment=left*), in the center (*alignment=center*) or to the right (*alignment=right*). As the default, multi-line label texts are formatted flush left. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(135 90 135)
  PieChart(label+oval;10;50)
  BorderStyle(all;none)
  LabelTexts(1;"Region A\n|u|\n(|f1|%)";
              "Region B\n|u|\n(|f1|%)";
              "Region C\n|u|\n(|f1|%)")
  LabelStyle(all;;;bold+italic;darkBlue;center)
CloseDrawing()
```



The 7th argument *orientation* makes it possible to turn texts in a desired direction. When entering an angle greater than zero, texts are turned clockwise; a negative angle turns them counterclockwise. All angles are to be entered within a range of  $\pm 360$  degrees; 0 degrees is horizontal (3 o'clock). Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(100 250 180 310 222 123 89 222)
  BarChart(label)
  LabelTexts(1;"|u|")
  LabelStyle;;;bold;;;-90)
  LabelOptions(1;out)
  GridLocation(xy;none) // hide grid
CloseDrawing()
```



When setting a maximum width (*maxWidth*>0), longer multi-line texts can be issued as continuous text with automatic word wraparound. The height of the text block can be controlled by the argument *maxHeight*. Texts which do not completely fit into the area predefined by *maxWidth* and *maxHeight* will be shortened according to the setting by the argument *ellipsisPosition*. Five options are available:

*ellipsisPosition*=0: Text is clipped at the end, no "..."

*ellipsisPosition*=1: Text is clipped in the beginning (...text)

*ellipsisPosition*=2: Text is clipped in the middle (text...text)

*ellipsisPosition*=3: Text is clipped at the end (text...) – default

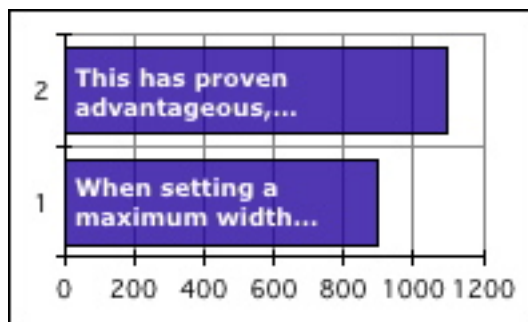
*ellipsisPosition*=4: Text is clipped in the beginning + at the end (...text...)

Examples:

```

OpenDrawing(200;120;;2)
  AddFrame(0;0;200;120)
  ChartData(900 1100)
  BarChart(label+horizontal;30)
  FillStyle(1;0 0 150 200)
  LabelStyle(1;"Verdana";;bold;white;left;;110;30;3)
  LabelOptions(;begin)
  LabelTexts(1;"When setting a maximum width longer
multi-line texts can be issued as continuous text with
automatic word wraparound." ; "This has proven advantageous,
particularly for Gantt charts such as used for project
management and date planners.")
  CloseDrawing()

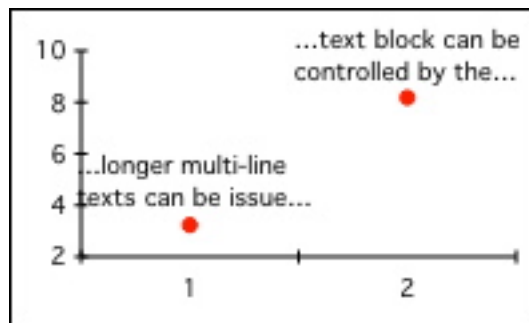
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3.23 8.18)
  ScatterChart(label;on)
  SymbolStyle(1;bullet;5)
  LabelStyle(1;;;;;left;;85;30;4)
  LabelOptions(1;topCenter)
  LabelTexts(1;"When setting a maximum width, longer
multi-line texts can be issued as continuous text with
automatic word wraparound." ; "The height of the text block
can be controlled by the argument maxHeight. ")
  GridLocation(xy;none)
CloseDrawing()

```

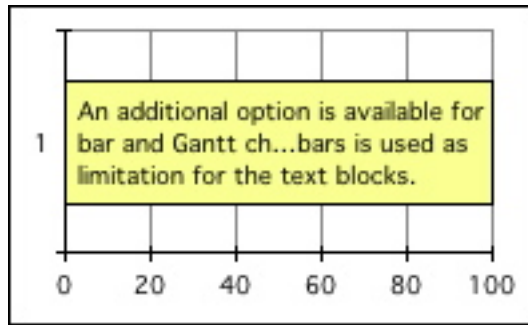


As the default, *maxWidth=-1* and *maxHeight=-1* are set, i.e. no width and height constraints and, thus, no automatic word wraparound support. An additional option is available for bar and Gantt charts. When *maxWidth=-2* and/or *maxHeight=-2*, the width and/or height of the bars is used as limitation for the text blocks. This has proven advantageous, particularly for Gantt charts such as used for project management and date planners. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(100 )
  BarChart(label+horizontal;80)
  FillStyle(1;lightYellow)
  LabelStyle(1;;;;;-2;-2;2)
  LabelTexts(1;"An additional option is available for
bar and Gantt charts. When maxWidth=-2 and/or maxHeight=-2,
the width and/or height of the bars is used as limitation for
the text blocks. ")
  CloseDrawing()

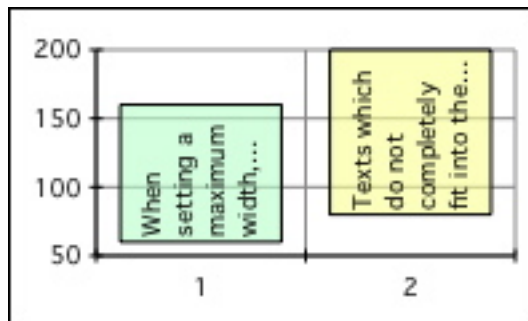
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(60 160;80 200)
  GanttChart(label+horizontal;30)
  FillStyle(1;200 255 200 150)
  FillStyle(2;255 255 150 150)
  LabelStyle(all;;;;;-90;-2;-2;3)
  LabelTexts(1;"When setting a maximum width, longer
multi-line texts can be issued as continuous text with
automatic word wraparound. The height of the text block can
be controlled by the argument maxHeight.")
  LabelTexts(2;"Texts which do not completely fit into
the area predefined by maxWidth and maxHeight will be shortened
according to the setting by the argument ellipsisPosition.")
  CloseDrawing()

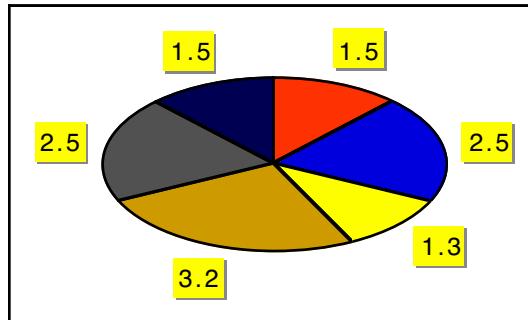
```



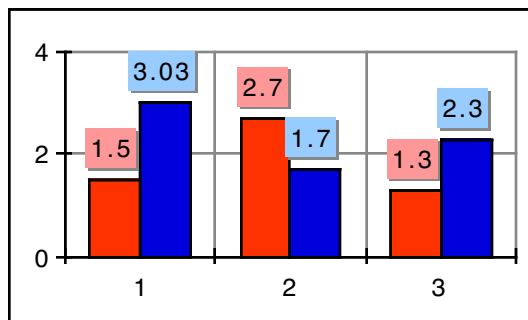
**LabelBackground(seriesIndex;fillColor;fillPattern;  
borderWidth;borderColor;borderPattern;  
shadowOffset;shadowColor;shadowPattern)**

By using the LabelBackground() function, a background can be added to labels. In doing so, it is possible to vary the color, pattern, border and shadow. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1.3 3.2 2.5 1.5)
  PieChart(oval+label)
  LabelBackground(1;yellow;;;0;;;1)
CloseDrawing()
```



```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.7 1.3; 3.03 1.7 2.3)
  BarChart(label)
  LabelBackground(1;lightRed;;;0;;;1)
  LabelBackground(2;lightBlue;;;0;;;1)
CloseDrawing()
```



**LabelOptions(seriesIndex;location;hOffset;vOffset;  
lowerLimit;upperLimit)**

The LabelOptions() function offers numerous ways to position and finely adjust labels:

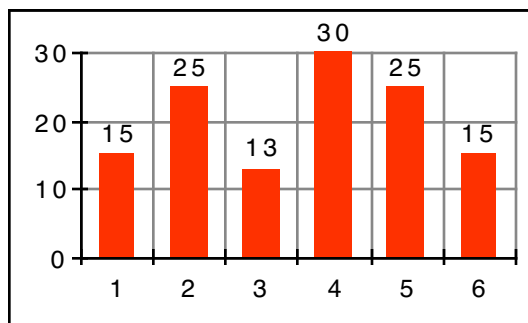
- *Location of the labels on non-stacked bar charts:*

If enough space is available, the labels are placed by default outside of the bars, otherwise on the inside. Using the argument *location*, nine positions can be defined on bar charts:

<i>constant</i>	<i>value</i>	<i>positioning</i>
smartBegin	1	beginning of bar if enough space is available
smartCenter	2	middle of bar if enough space is available
smartEnd	3	end of bar if enough space is available
begin	4	beginning of bar (forced)
center	5	middle of bar (forced)
end	6	end of bar (forced)
edge	7	edge on end of bar (forced)
smartOut	8	outside if enough space is available (default)
out	9	outside (forced)

Examples:

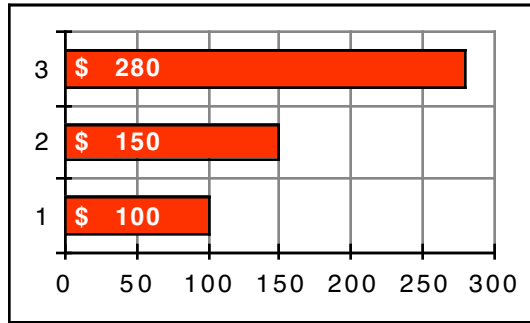
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(15 25 13 30 25 15)
  BarChart(label)
  BorderStyle(all;none) // hide border
  LabelOptions(1;out)
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(100 150 280)
  BarChart(label+horizontal)
  LabelTexts(1;"$ |u|")
  LabelStyle(;;;bold;white)
  LabelOptions(1;begin)
CloseDrawing()

```



More examples can be found in the section *Charts*.

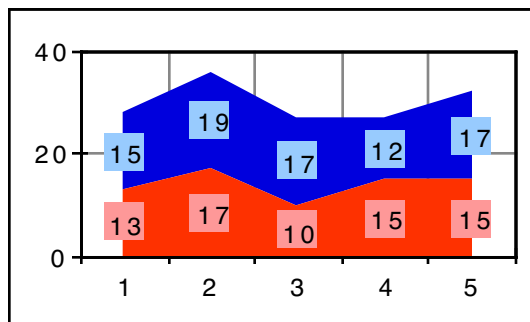
- *Location of the labels on stacked charts:*

On stacked and proportional bar and area charts the labels are placed in the center of each bar or area section by default; the argument *location* is ignored. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(13 17 10 15 15; 15 19 17 12 17)
  AreaChart(stacked+label;on)
  BorderStyle(all;none)
  LabelBackground(1;lightRed;;0)
  LabelBackground(2;lightBlue;;0)
  GridFrame()
CloseDrawing()

```

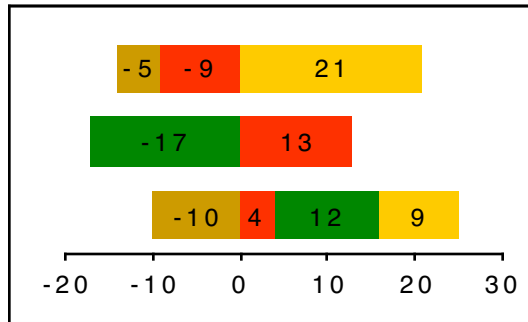




```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 -9;
            12 -17 0;
            9 0 21;
            -10 0 -5)
  BarChart(stacked+horizontal+label;50)
  BorderStyle(all;none)
  FillStyle(2;green)
  FillStyle(3;darkYellow)
  GridLocation(all;none) // hide grid
  AxisOptions(y;none)    // hide y-axis
  AxisMajorTicks(x;2;;;out)
CloseDrawing()

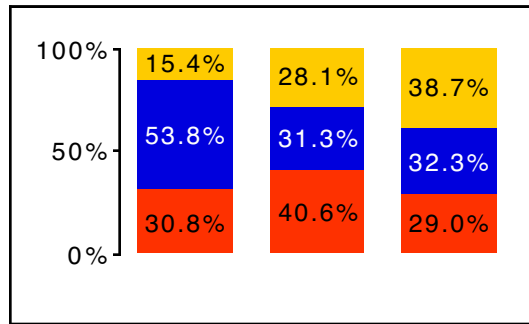
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 13 9;
            7 10 10;
            2 9 12)
  BarChart(proportional+label;40)
  Scaling(y;percent)
  BorderStyle(all;none) // hide borders
  FillStyle(3;darkYellow)
  LabelTexts(all;"|2f1|")
  LabelStyle(2;;;white)
  GridLocation(all;none) // hide grid
  AxisOptions(x;none)    // hide x-axis
  AxisMajorTicks(y;2;;;out)
CloseDrawing()

```



In addition, it is possible to represent either the total sums or the running totals. The drawing and positioning of (running) totals are controlled by the four style functions `LabelTexts()`, `LabelStyle()`, `LabelBackground()` and `LabelOptions()` in combination with `seriesIndex=-1` (`seriesIndex=stacked`). This special series index is available only in the four above-mentioned style functions in combination with stacked or proportional charts. The (running) totals can be placed either outside (default) or on the border of each individual bar or area section. The following four constants are available for the argument *location*:

<i>constant</i>	<i>value</i>	<i>positioning</i>
<code>totalsOut</code>	1	totals outside
<code>totalsEdge</code>	2	totals on the border
<code>runningTotalsOut</code>	3	running totals outside
<code>runningTotalsEdge</code>	4	running totals on the border

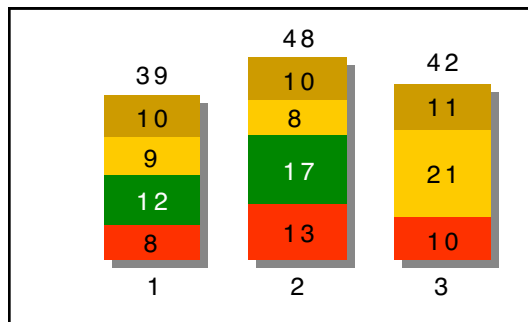
The other usual positioning constants are ignored when `seriesIndex=-1` or `seriesIndex=stacked`.

Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 8 13 10;
            12 17 0;
            9 8 21;
            10 10 11)
  BarChart(stacked+label+shadow;50)
  LabelOptions(stacked;totalsOut)
  BorderStyle(all;none)
  LabelStyle(2;;;white)
  FillStyle(2;green)
  FillStyle(3;darkYellow)
  GridLocation(all;none) // hide grid
  AxisOptions(y;none)    // hide y-axis
  AxisMajorTicks(x;0)    // hide x-axis tick marks
  AxisLine(x;0)          // hide x-axis line
CloseDrawing()

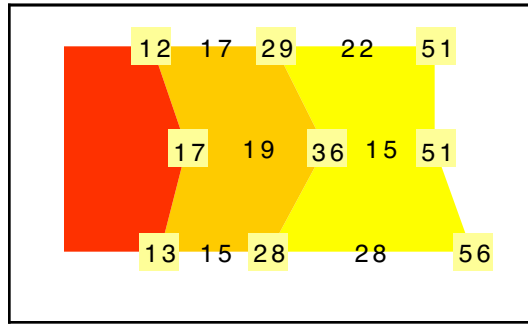
```



```

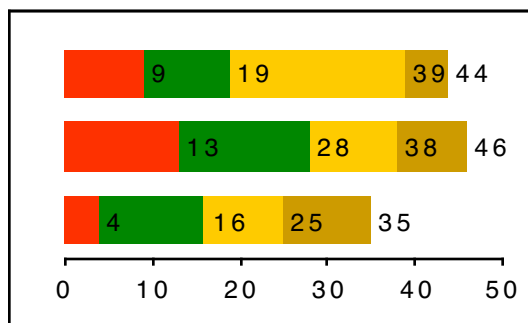
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(13 17 12;
            15 19 17;
            28 15 22)
  AreaChart(stacked+label+horizontal)
  LabelTexts(1;" ")
  BorderStyle(all;none)
  FillStyle(2;darkYellow)
  LabelOptions(-1;runningTotalsEdge)
  LabelBackground(stacked;lightYellow;;0)
  GridLocation(all;none) // hide grid
  AxisOptions(all;none)  // hide axes
CloseDrawing()

```



If only the total sums or the running totals are to be represented, the "default labels" can be suppressed by entering "blank" labels `LabelTexts(all;" ")` instead. Examples:

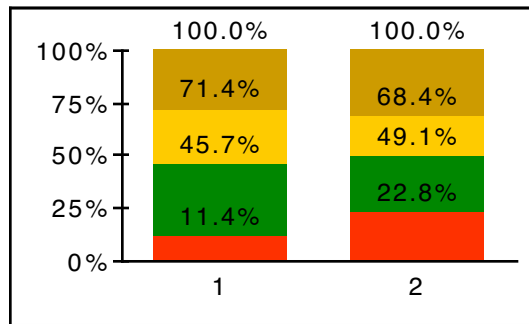
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 9;
            12 15 10;
            9 10 20;
            10 8 5)
  BarChart(stacked+horizontal+label;50)
  LabelTexts(all;" ")
  LabelOptions(stacked;runningTotalsOut)
  BorderStyle(all;none)
  FillStyle(2;green)
  FillStyle(3;darkYellow)
  GridLocation(all;none) // hide grid
  AxisOptions(y;none)    // hide y-axis
  AxisMajorTicks(x;2;;;out)
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13;
            12 15;
            9 11;
            10 18)
  BarChart(proportional+label;50)
  Scaling(y;percent;0;100;4)
  LabelTexts(all;" ")
  LabelTexts(stacked;"|2f1|%")
  LabelOptions(stacked;runningTotalsOut)
  BorderStyle(all;none)
  FillStyle(2;green)
  FillStyle(3;darkYellow)
  GridLocation(all;none) // hide grid
  AxisMajorTicks(x;0)
CloseDrawing()

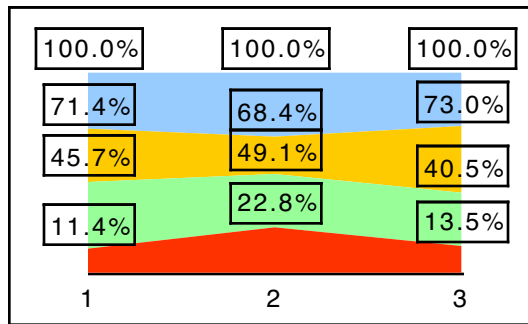
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  OpenChart(30;25;140;75;on)
  ChartData(4 13 5; 12 15 10; 9 11 12; 10 18 10)
  AreaChart(proportional+label)
  LabelTexts(all;" ")
  LabelTexts(stacked;"|2f1|%")
  LabelBackground(stacked;;transparent)
  LabelOptions(stacked;runningTotalsOut)
  BorderStyle(all;none)
  FillStyle(2;lightGreen)
  FillStyle(3;darkYellow)
  FillStyle(4;lightBlue)
  GridLocation(all;none)
  AxisOptions(y;none) // hide y-axis
  AxisMajorTicks(x;0)
CloseDrawing()

```



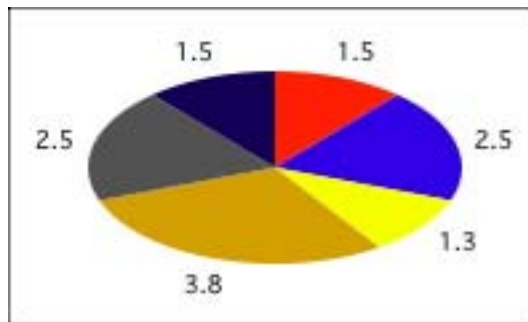
- *Location of the labels on pie charts:*

On pie charts the labels are always located outside of the chart — the argument *location* is ignored. However, by using the `PieChartInnerLabelTexts()` and `PieChartCenterLabelText()` functions it is also possible to position texts inside and in the center of pie charts. Detailed information, including examples, can be found in the *Charts* section. Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1.3 3.8 2.5 1.5)
  PieChart(oval+label)
  BorderStyle(all;none)
CloseDrawing()

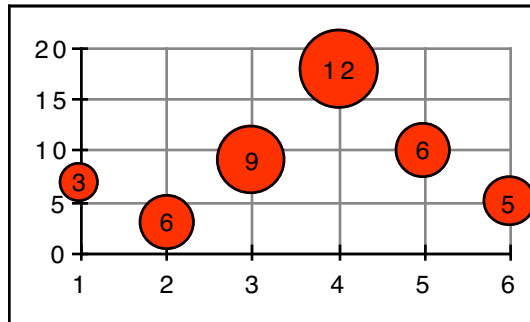
```



- *Location of the labels on bubble charts:*

As the default, the labels are positioned in the center (*location=center-Center*). Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 3 9 18 10 5; // y-values
            3 6 9 12 6 5) // diameters
  BubbleChart(label)
CloseDrawing()
```



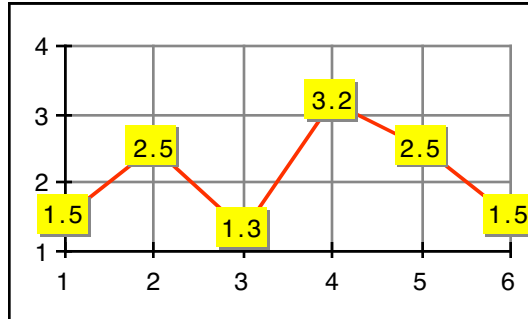
- *Location of the labels on all other charts:*

As the default, the labels are positioned on the right above the chart points. Using the argument *location* nine positions can be defined:

<i>constant</i>	<i>value</i>
topLeft	1
topCenter	2
topRight	3 (default)
centerLeft	4
centerCenter	5
centerRight	6
bottomLeft	7
bottomCenter	8
bottomRight	9

Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1.3 3.2 2.5 1.5)
  LineChart(label)
  LabelBackground(1;yellow;;0;;;1)
  LabelOptions(all;centerCenter)
CloseDrawing()
```

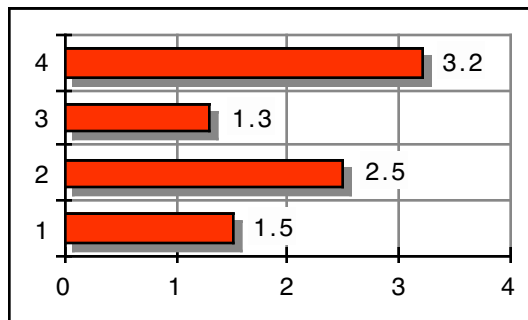


As an option, the location of the labels can be finely adjusted by using the arguments *hOffset* and *vOffset*. Positive offset values move the labels down to the right, negative values up to the left. Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1.3 3.2)
  BarChart(label+horizontal+shadow)
  LabelBackground(1;white;;0)
  LabelOptions(1;;3) // 3 pixels to the right
CloseDrawing()

```



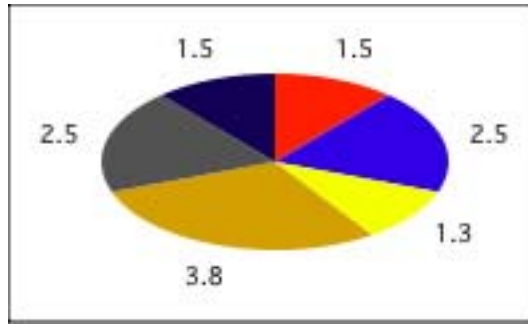
On pie charts the arguments *hOffset* and *vOffset* are ignored; instead, a radial offset value can be defined by using the *PieChartLabelOptions(;outerLabelOffset)*. A positive offset value increases the space between the border and labels; a negative value decreases the space. The argument *outerLabelOffset* is to be entered in percent of the segment length. Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1.3 3.8 2.5 1.5)
  PieChart(oval+label)
  PieChartLabelOptions(;10)
  BorderStyle(all;none)
CloseDrawing()

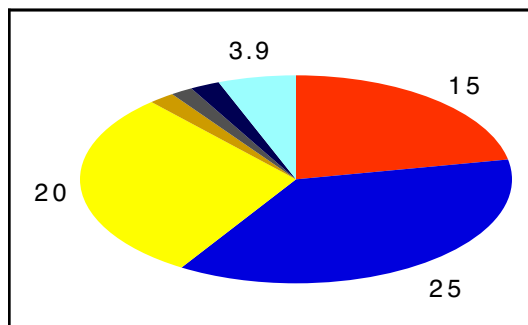
```





With the aid of the arguments *lowerLimit* and *upperLimit* the labeling of chart values can be suppressed, i.e. only chart values larger than the *lowerLimit* and smaller than the *upperLimit* are labeled. For example, by entering a lower limit the labeling of narrow pie chart segments can be prevented. Example:

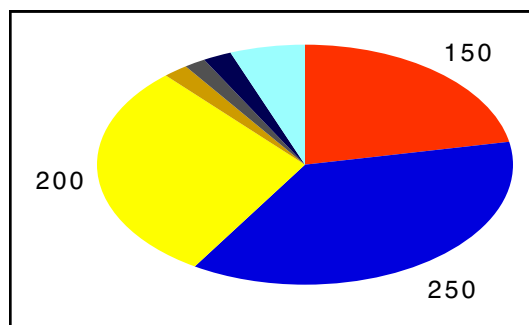
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(15 25 20 1.3 1.2 1.5 3.9)
  PieChart(label+oval)
  BorderStyle(all;none)
  LabelOptions(1;;0;0;3.0)
CloseDrawing()
```



On pie charts it is also possible to choose between absolute and relative limits by using the `PieChartLabelOptions(useRelativeLimits)` function. Relative limits are to be entered in percent of the total sum.

Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(150 250 200 13 12 15 39)
  PieChart(label+oval)
  BorderStyle(all;none) // hide borders.
  // hide label if value is less
  // than 10% of the total sum.
  LabelOptions(1;;0;0;10.0)
  PieChartLabelOptions(on)
CloseDrawing()
```



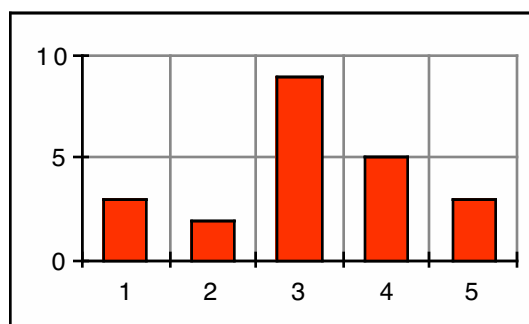
## Background

A total of four functions are available for designing the background; two for creating the "entire background", i.e. the background covers the entire drawing or the surrounding view, and two other functions for creating the actual chart background, i.e. the background is limited to the grid area.

**Background(fillColor;fillPattern;borderWidth;  
borderColor;borderPattern;shadowOffset;  
shadowColor;shadowPattern)**

The Background() function makes it possible to define a single-color background including border and shadow. If the Background() function is entered without arguments, a white background with a 1-pixel wide black border without shadow is drawn. Example:

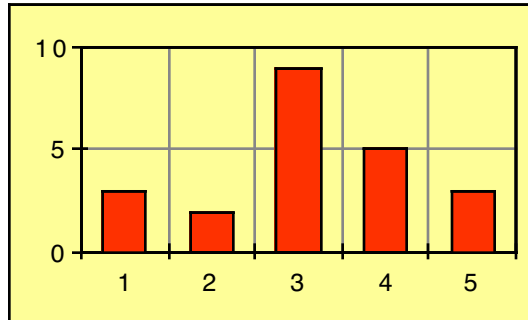
```
OpenDrawing(200;120)
  ChartData(3 2 9 5 3)
  BarChart()
  Background()
CloseDrawing()
```



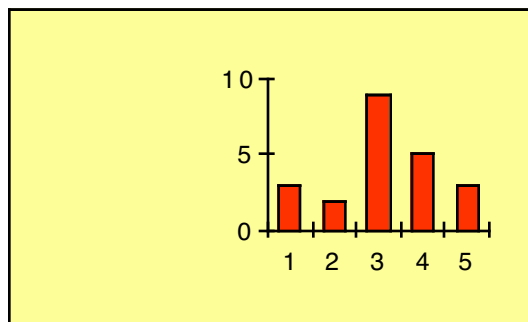
If no views are used or if the background function is entered outside of the views, then the function is applied to the entire drawing. Views are explained in the *Layout* section.

Examples:

```
OpenDrawing(200;120)
  ChartData(3 2 9 5 3)
  BarChart()
  GridFrame()
  Background(lightYellow)
CloseDrawing()
```



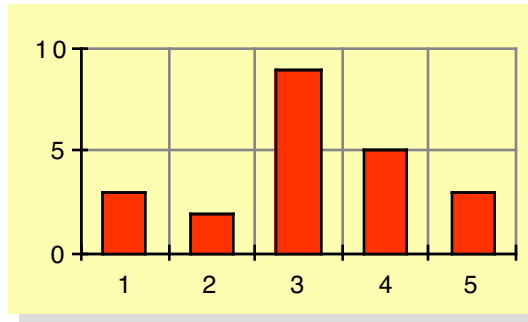
```
OpenDrawing(200;120)
  OpenView(70;10;120;100)
    ChartData(3 2 9 5 3)
    BarChart()
    GridLocation(all;none) // hide grid
  CloseView()
  Background(lightYellow)
CloseDrawing()
```



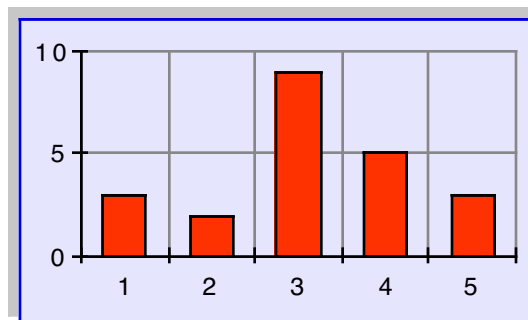
If the shadow offset is positive, then the shadow will be drawn on the bottom right; if the offset is negative, the shadow will be on the top left.

Examples:

```
OpenDrawing(200;120)
  ChartData(3 2 9 5 3)
  BarChart()
  Background(255 255 180;;;0;;;4;lightGray)
CloseDrawing()
```

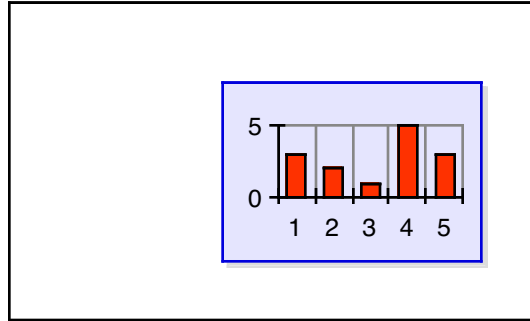


```
OpenDrawing(200;120)
  ChartData(3 2 9 5 3)
  BarChart()
  Background(230 230 255;;;1;blue;;;-4;200 200 200)
CloseDrawing()
```



If the Background( ) function is entered within a view, then the function is applied to the background of the view. Examples:

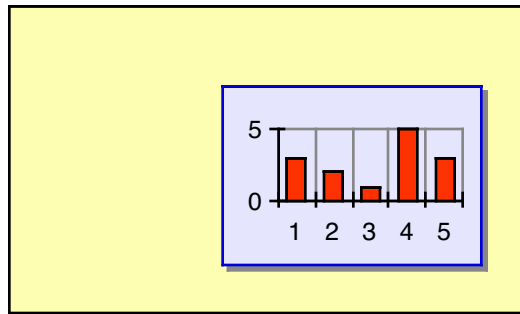
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  OpenView(80;30;100;70)
  AddFrame(0;0;100;70)
  ChartData(3 2 1 5 3)
  BarChart()
  Background(230 230 255;;;1;blue;;;2;lightGray)
  CloseView()
CloseDrawing()
```



```

OpenDrawing(200;120)
  OpenView(80;30;100;70)
    AddFrame(0;0;100;70)
    ChartData(3 2 1 5 3)
    BarChart()
    Background(230 230 255;;1;blue;;2)
  CloseView()
  Background(255 255 180;;1;;;4;lightGray)
CloseDrawing()

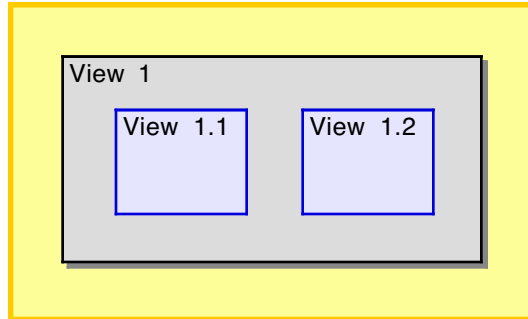
```



```

OpenDrawing(200;120)
  OpenView(20;20;160;80)
    Background(lightGray;;1;;;2)
    AddText(3;10;"View 1")
    OpenView(20;20;50;40)
      Background(230 230 255;;1;blue)
      AddText(3;10;"View 1.1")
    CloseView()
    OpenView(90;20;50;40)
      Background(230 230 255;;1;blue)
      AddText(3;10;"View 1.2")
    CloseView()
  CloseView()
  Background(lightYellow;;2;darkYellow)
CloseDrawing()

```



**BackgroundPict(*sourceType*;*sourceName*;*location*;  
adjustment;*isProportional*)**

The BackgroundPict() function makes it possible to use a picture as the background. Three different picture sources are supported:

- *Clipboard*: (*sourceType*=*clipboard*)

The clipboard is used when the constant *clipboard* is entered as the *sourceType* or the argument *sourceType* remains empty. The 2nd argument *sourceName* has no meaning and is ignored.

Examples: BackgroundPict(*clipboard*)

BackgroundPict()

- *File*: (*sourceType*=*file*)

Either a complete, absolute file path or only a relative path can be passed. The relative path refers to the folder in which the current FileMaker Pro database file is located. Examples:

BackgroundPict(*file*;"Pictures/Pict\_01.png")

BackgroundPict(*file*;"C:/Pictures/Pict\_01.png")

BackgroundPict(*file*;"Macintosh HD/Picts/Pict\_01.pdf")

In Mac OS X pictures have to be in PDF, PICT, GIF, JPEG, PNG, BMP, TIFF format in order to be imported.

In Windows pictures have to be in WMF, EMF, GIF, JPEG, PNG, BMP, TIFF format in order to be imported.

- *Resource*: (*sourceType*=*resource*)

Presently there are a total of 42 gradient backgrounds stored in resources. These can be accessed by entering an index number between 1 and 42. Please note that the index number is to be placed in double quotes. An overview of the predefined backgrounds can be found in *xmReference*.

Furthermore, when using resources in Windows, please note that charts can grow to a size consisting of several MB. This problem only arises in connection with the standard output format EMF and not with the bitmap output format. EMF and bitmap format are explained along with the OpenDrawing() function in the *Layout* section.

The 3rd argument *location* provides nine different positions for the location of the picture in the background.

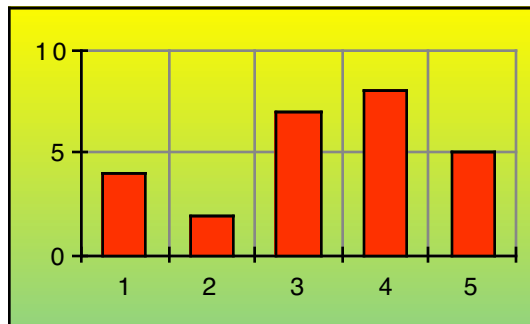
<i>constant</i>	<i>value</i>
topLeft	1
topCenter	2
topRight	3
centerLeft	4
centerCenter	5 (default)
centerRight	6
bottomLeft	7
bottomCenter	8
bottomRight	9

The 4th argument *adjustment* makes five options available for adjusting the picture in the background.

<i>constant</i>	<i>value</i>
crop	1
reduce	2
enlarge	3
reduceOrEnlarge	4 (default)
tile	5

In addition, by using the 5th argument *isProportional*, it is possible to define whether the picture width to picture height ratio should remain unchanged when adjusting it to the background area. When *adjustment=crop* or *adjustment=tile*, the argument *isProportional* is ignored. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 2 7 8 5)
  BarChart()
  BackgroundPict(resource;"7")
CloseDrawing()
```

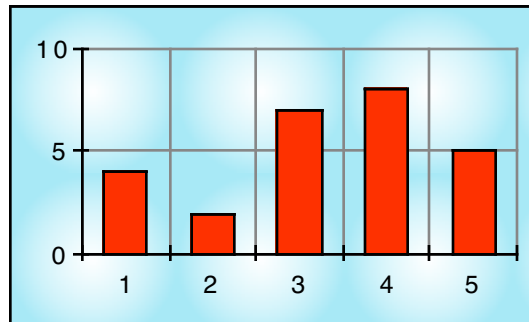




```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 2 7 8 5)
  BarChart()
  BackgroundPict(resource;"36";;tile)
CloseDrawing()

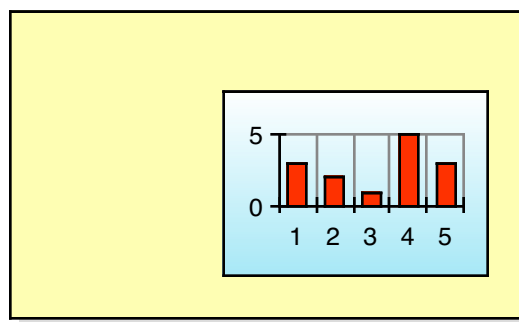
```



```

OpenDrawing(200;120)
  OpenView(80;30;100;70)
  AddFrame(0;0;100;70)
  ChartData(3 2 1 5 3)
  BarChart()
  BackgroundPict(resource;"3")
  CloseView()
  Background(255 255 180;;;1;;;4;lightGray)
CloseDrawing()

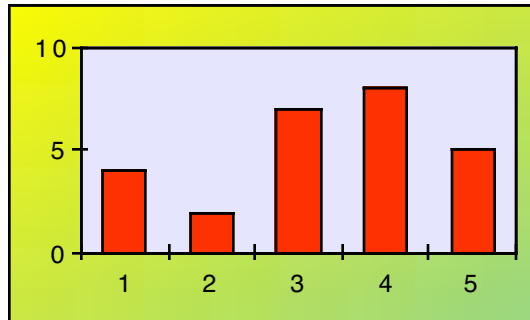
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 2 7 8 5)
  BarChart()
  GridLocation(all;none) // hide grid
  ChartBackground(all;230 230 255)
  BackgroundPict(resource;"29")
CloseDrawing()

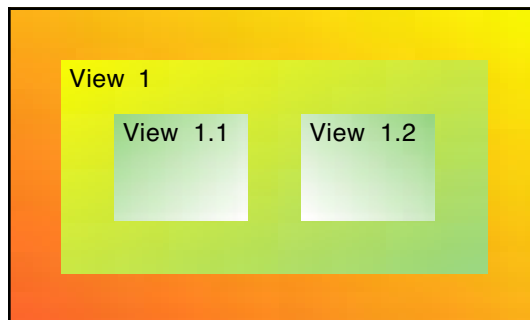
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  OpenView(20;20;160;80)
    BackgroundPict(resource;"29")
    AddText(3;10;"View 1")
    OpenView(20;20;50;40)
      BackgroundPict(resource;"26")
      AddText(3;10;"View 1.1")
    CloseView()
    OpenView(90;20;50;40)
      BackgroundPict(resource;"28")
      AddText(3;10;"View 1.2")
    CloseView()
  CloseView()
  BackgroundPict(resource;"32")
CloseDrawing()

```



Furthermore, it is possible to combine background functions with functions such as `AddFrame()`. Example:

```
OpenDrawing(200;120)
  ChartData(4 2 7 8 5)
  BarChart()
  BackgroundPict(resource;"27")
  AddFrame(1;1;198;118;2;green)
CloseDrawing()
```

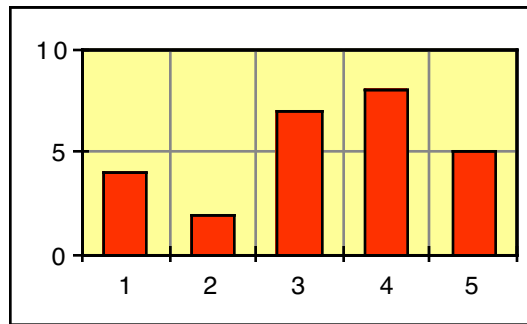


**ChartBackground(planeIndex;fillColor;fillPattern;  
borderWidth;borderColor;borderPattern;  
shadowOffset;shadowColor;shadowPattern)**

The `ChartBackground()` function is set up exactly the same way as the `Background()` function except for the 1st argument *planeIndex*. Unlike the `Background()` function, the background area for the `ChartBackground()` function is limited to the grid area of a chart. As the 1st argument *planeIndex*, the constants *xy* or *all* are to be entered for 2-dimensional charts. In addition, for 3-dimensional bar and Gantt charts the plane constants *xz* and *yz* are available. The z-axis points in the direction of the viewer.

Examples:

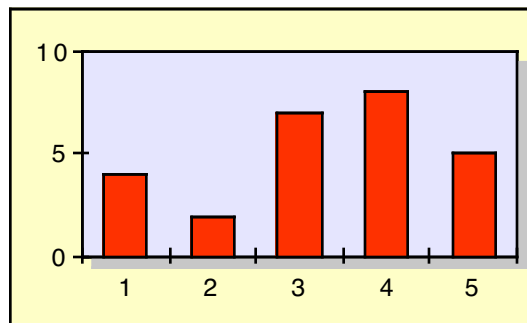
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 2 7 8 5)
  BarChart()
  GridFrame()
  ChartBackground(xy;lightYellow)
CloseDrawing()
```



```

OpenDrawing(200;120)
  ChartData(4 2 7 8 5)
  BarChart()
  GridLocation(all;none) // hide grid
  ChartBackground(;230 230 255;;1;black;;4;200 200 200)
  Background(255 255 200)
CloseDrawing()

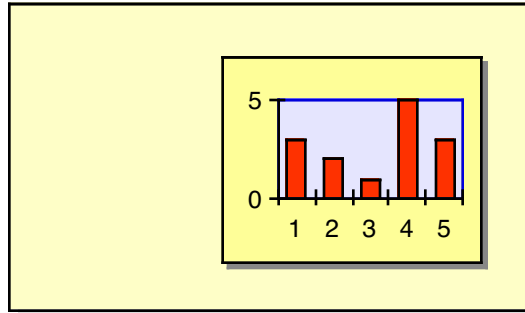
```



```

OpenDrawing(200;120)
  OpenView(80;20;100;80)
  ChartData(3 2 1 5 3)
  BarChart()
  GridLocation(all;none) // hide grid
  ChartBackground(xy;230 230 255;;1;blue)
  Background(lightYellow;;;;2)
  CloseView()
  Background(255 255 200;;1;;;4;lightGray)
CloseDrawing()

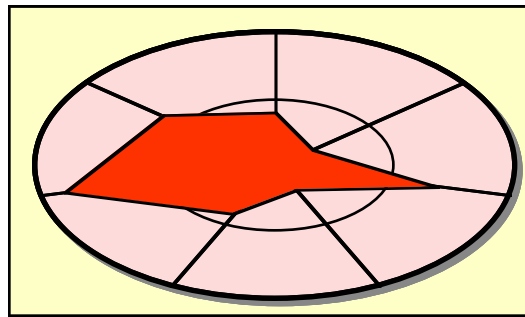
```



```

OpenDrawing(200;120)
  ChartData(4 2 7 2 4 9 6)
  RadarChart(oval)
  RadarChartOptions(0)
  MajorGridLineColors(all;all;black)
  ChartBackground(xy;255 220 220;;2;black;;2)
  Background(255 255 200;;1;;;3;lightGray)
CloseDrawing()

```



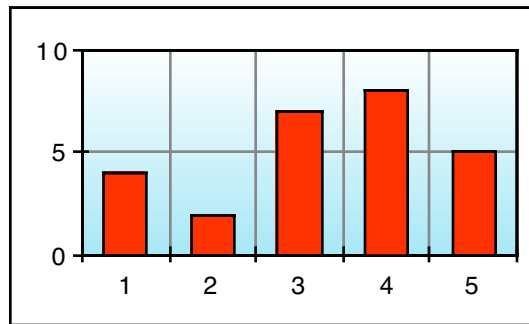
### **ChartBackgroundPict(planeIndex;sourceType;sourceName)**

By using the `ChartBackgroundPict()` function, a picture background limited to the grid area can be defined. The 1st argument *planeIndex* is identical to the one in the `ChartBackground()` function, the other arguments, *sourceType* and *sourceName*, are described in combination with the `BackgroundPict()` function. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 2 7 8 5)
  BarChart()
  GridFrame()
  ChartBackgroundPict(all;resource;"3")
CloseDrawing()

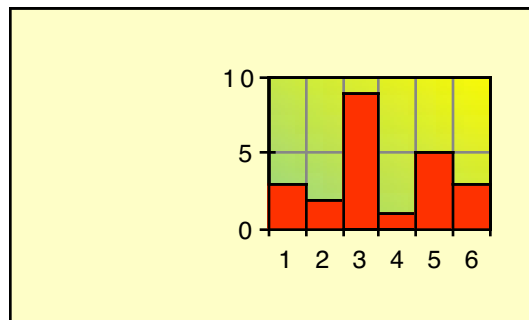
```



```

OpenDrawing(200;120)
  OpenView(70;10;120;100)
    ChartData(3 2 9 1 5 3)
    BarChart(;0)
    GridFrame()
    ChartBackgroundPict(xy;resource;"30")
  CloseView()
  Background(255 255 200)
CloseDrawing()

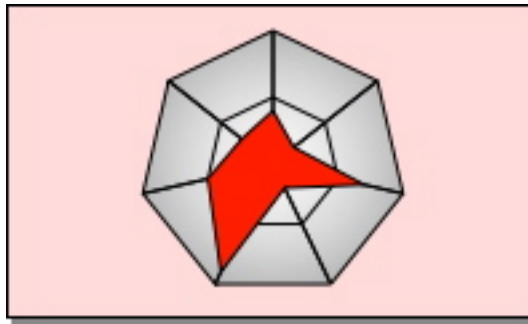
```



```

OpenDrawing(200;120)
  ChartData(4 2 7 2 9 5 3)
  RadarChart()
  RadarChartOptions(0;poly)
  MajorGridLineColors(all;all;black)
  ChartBackgroundPict(xy;resource;"34")
  Background(255 220 220;;1;black;;2;gray)
CloseDrawing()

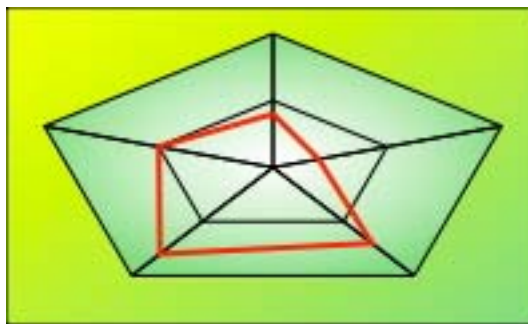
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 2 7 8 5)
  RadarChart(oval)
  RadarChartOptions(0;poly)
  FillStyle(all;;transparent)
  BorderStyle(1;poly;2;red)
  MajorGridLineColors(all;all;black)
  ChartBackgroundPict(xy;resource;"38")
  BackgroundPict(resource;"29")
CloseDrawing()

```



## Grids

A total of 12 functions are available for creating and designing grids. They make it possible to create:

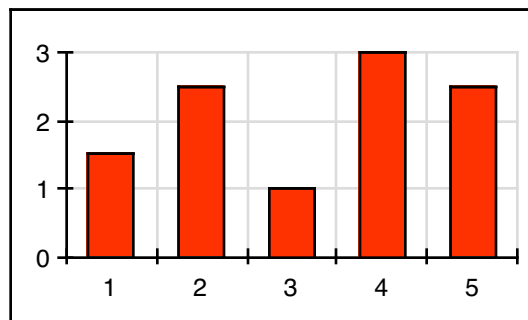
- grid lines and grid stripes
- major and minor grids
- a grid frame

1-pixel wide, gray grid lines are used as the default.

```
MajorGridLineWidths(directionAxis;distributionAxis;  
width1;width2...)  
MajorGridLineColors(directionAxis;distributionAxis;  
color1;color2...)  
MajorGridLinePatterns(directionAxis;distributionAxis;  
pattern1;pattern2...)
```

Grid lines can be defined differently with regard to color, pattern and line width. If the number of grid lines is larger than the number of defined line widths, colors or patterns, the latter will be repeated periodically. Example:

```
OpenDrawing(200;120)  
  AddFrame(0;0;200;120)  
  ChartData(1.5 2.5 1 3 2.5)  
  BarChart()  
  MajorGridLineColors(all;all;lightGray)  
CloseDrawing()
```



Grid lines can be suppressed by defining the line width as zero. If neither the directional axis nor the distribution axis is defined, the functions refer to all grid lines, i.e. for 2-dimensional Cartesian charts to



both the horizontal and vertical grid lines.

Access to a special grid direction is obtained by using both arguments *directionAxis* and *distributionAxis*.

2-dimensional, Cartesian charts:

horizontal grid lines: direction axis = x, distribution axis = y

vertical grid lines: direction axis = y, distribution axis = x

2-dimensional, polar charts:

radial grid lines: direction axis = x, distribution axis = y

concentric grid lines: direction axis = y, distribution axis = x

In addition, for 3-dimensional bar and Gantt charts the axis constant z is available. The z-axis points in the direction of the viewer.

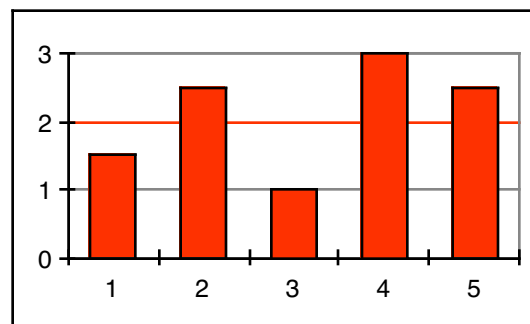
Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1 3 2.5)
  BarChart()

  // horizontal grid lines
  MajorGridLineColors(x;y;gray;gray;red)

  // hide vertical grid lines
  MajorGridLineWidths(y;x;0)

  // 1 pixel wide gray border
  GridFrame(xy;1;gray)
CloseDrawing()
```



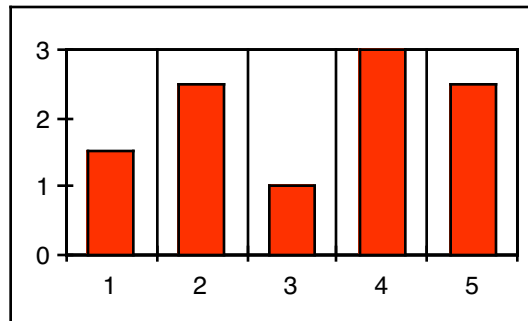
```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1 3 2.5)
  BarChart()
  MajorGridLineColors(all;all;black)

  // hide horizontal grid lines
  MajorGridLineWidths(x;y;0)

  // 1 pixel wide black border
  GridFrame()
CloseDrawing()

```



```

MinorGridLineWidths(directionAxis;distributionAxis;
width1;width2...)
MinorGridLineColors(directionAxis;distributionAxis;
color1;color2...)
MinorGridLinePatterns(directionAxis;distributionAxis;
pattern1;pattern2...)

```

The minor grid functions work the same way as the major grid functions. Please note that the minor grid can only be activated by defining a minor interval number greater than 1 in the `Scaling()` function (see section Axes).

## Examples:

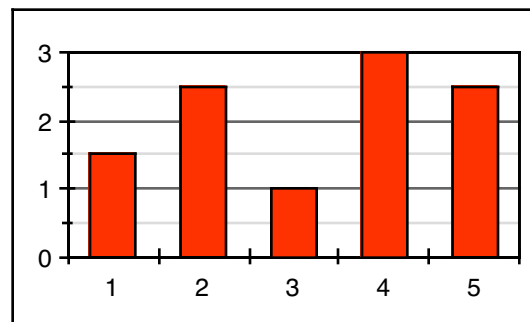
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1 3 2.5)
  BarChart()

  // 3 major intervals, 2 minor intervals
  Scaling(y;linear;;;3;2)

  // major grid
  MajorGridLineColors(all;all;darkGray)
  // hide vertical grid lines
  MajorGridLineWidths(y;x;0)

  // minor grid
  MinorGridLineColors(all;all;lightGray)

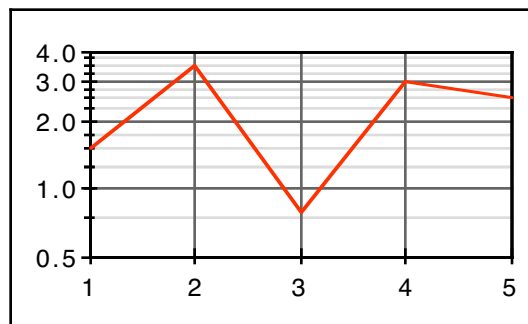
  // 1 pixel wide black frame
  GridFrame()
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 3.5 0.8 3 2.5)
  LineChart()
  Scaling(y;log;;;3;4)
  // major grid
  MajorGridLineColors(all;all;darkGray)
  // minor grid
  MinorGridLineColors(all;all;lightGray)
  // 1 pixel wide black frame
  GridFrame()
CloseDrawing()

```



Furthermore, a dash pattern can be assigned to the grid line width by adding a list of dash lengths and gaps. Example:

```

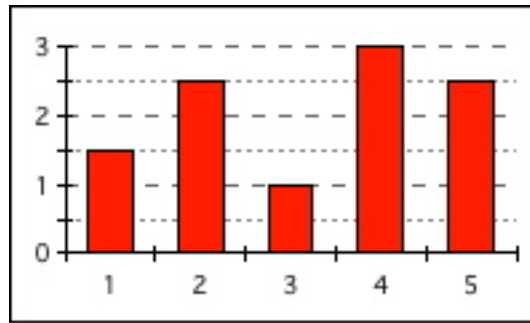
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1 3 2.5)
  BarChart()

  // 3 major intervals, 2 minor intervals
  Scaling(y;linear;;;3;2)

  // major grid lines
  MajorGridLineColors(x;y;darkGray)
  MajorGridLineWidths(x;y;1 5 5)
  // hide vertical grid lines
  MajorGridLineWidths(y;x;0)

  // minor grid lines
  MinorGridLineColors(x;y;gray)
  MinorGridLineWidths(x;y;1 2 2)
CloseDrawing()

```



More dash pattern examples can be found under `LineStyle()`.

```
MajorGridStripeColors(directionAxis;distributionAxis;  
                          color1;color2...)  
MajorGridStripePatterns(directionAxis;distributionAxis;  
                          pattern1;pattern2...)  
MinorGridStripeColors(directionAxis;distributionAxis;  
                          color1;color2...)  
MinorGridStripePatterns(directionAxis;distributionAxis;  
                          pattern1;pattern2...)
```

As an option, grid stripes can be used instead of grid lines or also in combination with grid lines. The setup of the grid stripe functions is the same as for the grid line functions.

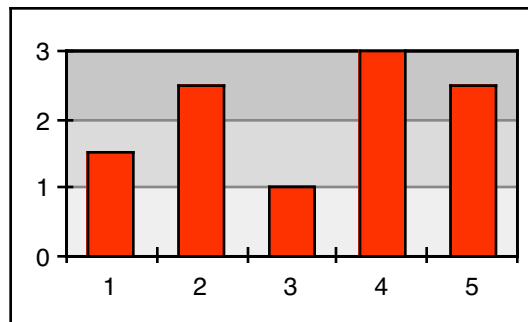
## Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1 3 2.5)
  BarChart()
  // horizontal grid stripes
  MajorGridStripeColors(x;y;240 240 240;
                        220 220 220;
                        200 200 200)

  // hide vertical grid lines
  MajorGridLineWidths(y;x;0)
  // 1 pixel wide black frame
  GridFrame()
CloseDrawing()

```

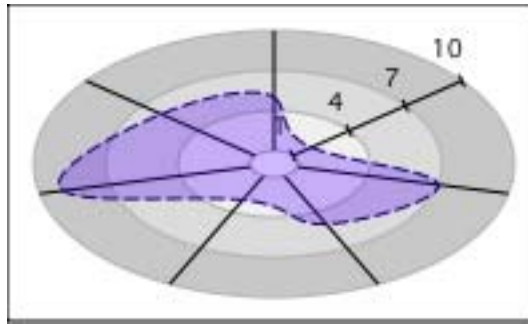


```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 2 7 5 3 9 6)
  RadarChart(oval)
  RadarChartOptions(2)
  Scaling(x;linear;1;10;3)
  FillStyle(1;50 0 255 70)
  BorderStyle(1;smooth;1 6 2;darkBlue)

  // grid
  MajorGridLineWidths(x;y;0)
  MajorGridLineColors(y;x;180 180 180)
  MajorGridStripeColors(y;x;240 240 240;
                        220 220 220;
                        200 200 200)
CloseDrawing()

```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1 3 2.5)
  BarChart(horizontal)
  Scaling(x;linear;0;3;3;4)

  // hide tick marks
  AxisMajorTicks(all;0)
  AxisMinorTicks(all;0)

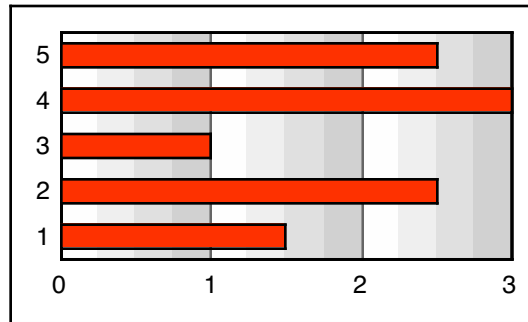
  // vertical grid lines
  MajorGridLineColors(y;x;darkGray)

  // vertical grid stripes
  MinorGridStripeColors(y;x;255 255 255;
                        240 240 240;
                        225 225 225;
                        210 210 210)

  // hide grid lines
  MajorGridLineWidths(x;y;0)
  MinorGridLineWidths(y;x;0)

  // 1 pixel wide black frame
  GridFrame()
CloseDrawing()

```

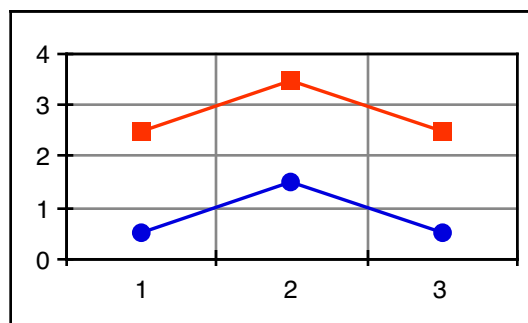


### **GridFrame(planeIndex;width;color;pattern)**

By using the GridFrame() function it is possible to add a frame around the grid whose width, color and pattern can be varied. As the 1st argument, *planeIndex*, the constant *xy* or *all* is to be entered for 2-dimensional charts. In addition, for 3-dimensional bar and Gantt charts the plane constants *xz* and *yz* are available. The z-axis points in the direction of the viewer.

Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(2.5 3.5 2.5; 0.5 1.5 0.5)
  LineChart(symbol;on)
  SymbolStyle(1;square;6)
  SymbolStyle(2;bullet;6)
  // 1 pixel wide black frame
  GridFrame(xy;1;black)
CloseDrawing()
```

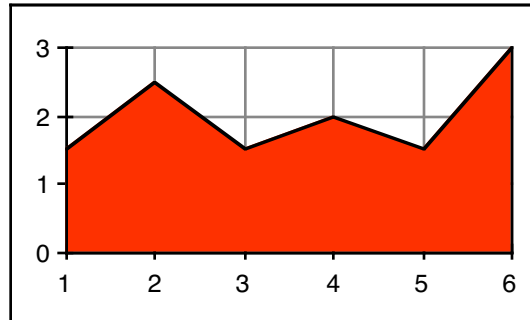




**GridLocation(planeIndex;gridLocation)**

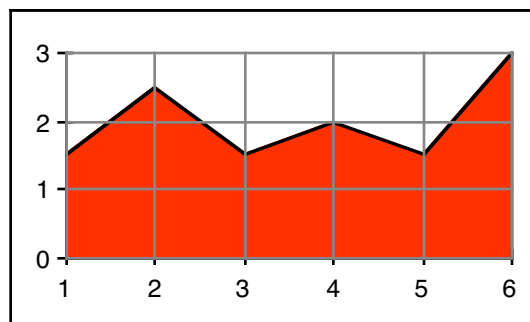
As the 1st argument *planeIndex* the constant *xy* or *all* is to be entered for 2-dimensional charts. In addition, for 3-dimensional bar and Gantt charts the plane constants *xz* and *yz* are available. The z-axis points in the direction of the viewer. As the default, the grid is always in the background of a chart (*gridLocation=back*). Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1.5 2 1.5 3)
  AreaChart()
  GridLocation(xy;back) // default location
CloseDrawing()
```



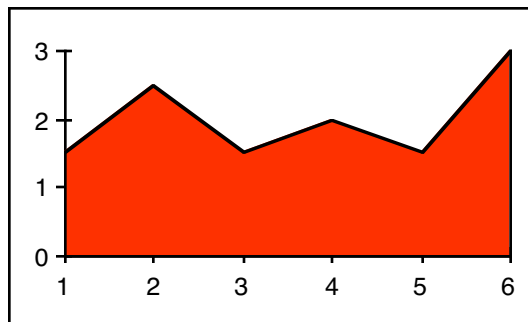
By using *gridLocation=front* the grid can be positioned in the foreground. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1.5 2 1.5 3)
  AreaChart()
  GridLocation(xy;front)
CloseDrawing()
```



In addition, all grids can be suppressed simply by using *gridLocation=none*.  
Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1.5 2.5 1.5 2 1.5 3)
  AreaChart()
  GridLocation(xy;none) // hide grids
CloseDrawing()
```



## Axes

A total of 18 functions are available for designing axes. They make it possible to:

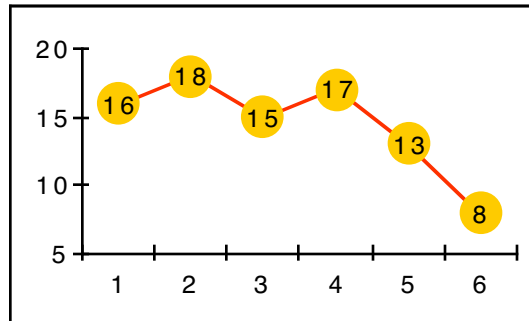
- define different types of scalings
- design axis lines, tick marks and scaling labels
- flexibly design and position axis labels
- make fine adjustments by using numerous options

For all 18 functions the axis index is defined as the 1st argument. The four constants *x*, *y*, *z* and *all* are available for this. If no axis index is defined or if *axisIndex=all* is set, then the respective function applies to all axes, e.g. to the x and y-axis on two-dimensional charts.

**Scaling(axisIndex;type;minValue;maxValue;  
numOfMajorIntervals;numOfMinorIntervals;  
logBaseValue;useEquidistantLogScaling)**

The type of scaling as well as the number of subdivisions and the scaling range can be defined by using the function `Scaling()`. If no scaling function is used, a suitable linear scaling is automatically computed. The 2nd argument *type* makes it possible to choose between linear (*type=linear*), percent (*type=percent*) and logarithmic scaling (*type=log*). If no type is defined, then linear scaling is used. Examples:

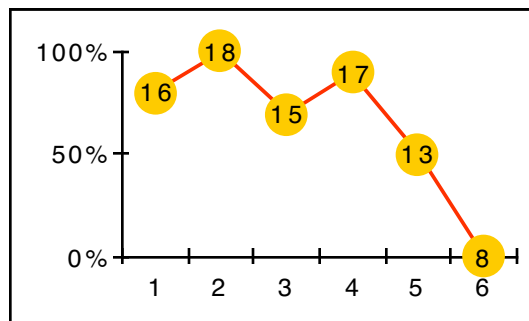
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(16 18 15 17 13 8)
  LineChart(symbol+label;on)
  Scaling(y;linear) // identical to automatic scaling
  SymbolStyle(1;bullet;15;;darkYellow)
  LabelOptions(all;centerCenter)
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(16 18 15 17 13 8)
  LineChart(symbol+label;on)
  Scaling(y;percent)
  SymbolStyle(1;bullet;15;;darkYellow)
  LabelOptions(all;centerCenter)
  GridLocation(all;none) // hide grid
CloseDrawing()

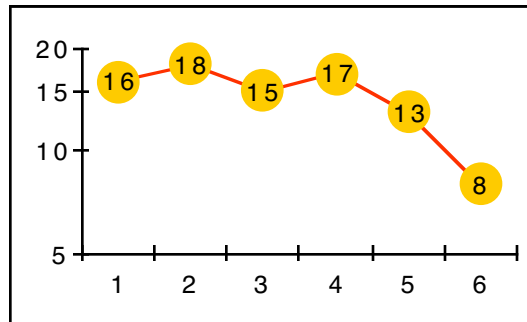
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(16 18 15 17 13 8)
  LineChart(symbol+label;on)
  Scaling(y;log)
  SymbolStyle(1;bullet;15;;darkYellow)
  LabelOptions(all;centerCenter)
  GridLocation(all;none) // hide grid
CloseDrawing()

```

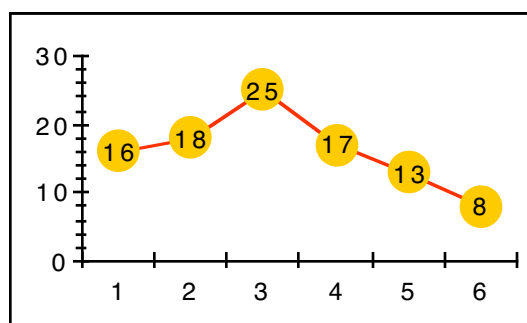


By using the 3rd and 4th argument, a minimum and maximum scaling value can be defined. If the minimum or maximum value is not defined, a suitable value is automatically sought.

The number of major and minor intervals can be controlled by using the 5th and 6th argument. The number of minor intervals applies to the number of intervals within a major interval, e.g. if the major intervals are to be subdivided into five minor intervals, *numOfMinorIntervals=5* should be set; if no minor tick marks are to be used, *numOfMinorIntervals=1* (default) should be set.

If no major interval is defined, the scaling is subdivided into four major intervals as the default. Examples:

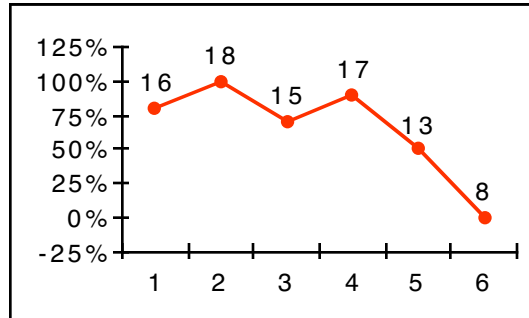
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(16 18 25 17 13 8)
  LineChart(symbol+label;on)
  // 3 major and 5 minor intervals
  Scaling(y;linear;0;30;3;5)
  SymbolStyle(1;bullet;15;;darkYellow)
  LabelOptions(all;centerCenter)
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(16 18 15 17 13 8)
  LineChart(symbol+label;on)
  Scaling(y;percent;-25;125;6)
  SymbolStyle(1;bullet;4;;red)
  LabelOptions(all;topCenter)
  GridLocation(all;none) // hide grid
CloseDrawing()

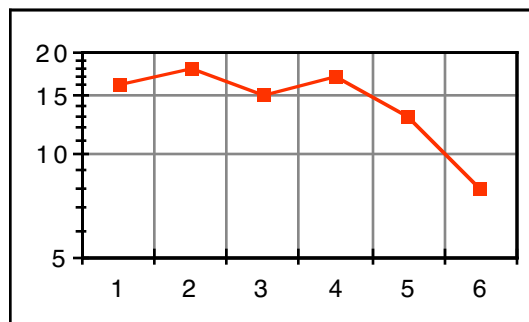
```



```

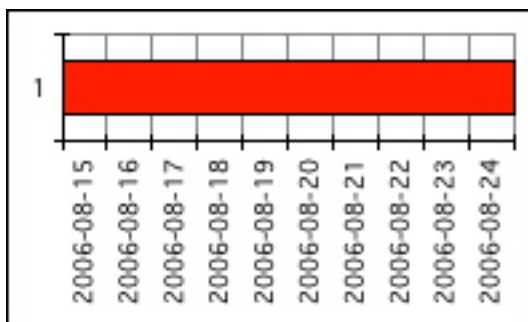
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(16 18 15 17 13 8)
  LineChart(symbol;on)
  // 3 major and 5 minor intervals
  Scaling(y;log;5;20;3;5)
  SymbolStyle(1;square;4;;red)
  MinorGridLineWidths(all;all;0) // hide minor grid
  GridFrame()
CloseDrawing()

```

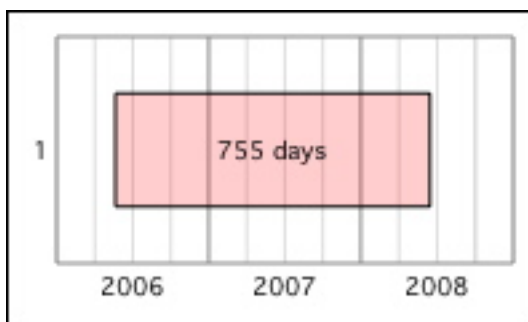


Dates and/or times can also be used as the minimum and maximum value (time axis). More information can be found in the *Data* section, *Entry of Date and Time* chapter. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(2006-08-15 2006-08-24)
  GanttChart()
  Scaling(x;linear)
  AxisMajorTickLabelStyle(x;;;;;-90)
CloseDrawing()
```



```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(2006-05-22 2008-06-14)
  GanttChart(label)
  FillStyle(1;255 0 0 50)
  Labeltexts(1;"|u| days")
  Scaling(x;linear;2006/1/1;2008/12/31;;4)
  AxisLine(all;0)
  AxisMajorTicks(all;0)
  AxisMinorTicks(all;0)
  AxisMajorTickLabelTexts(x;"|YYYY|")
CloseDrawing()
```



In the event of a time axis major and minor intervals can be defined by the following scaling constants.

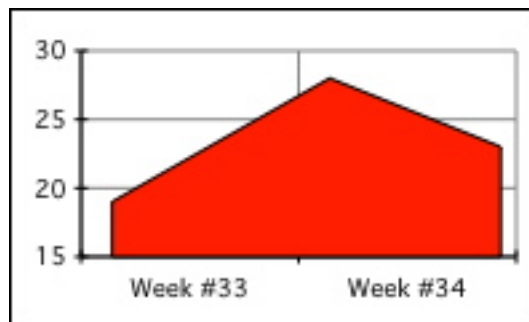
<i>constant</i>	<i>value</i>
year	-1
quarter	-2
month	-3
week	-4
day	-5
hour	-6
minute30	-7
minute20	-8
minute15	-9
minute10	-10
minute5	-11
minute	-12
second30	-13
second20	-14
second15	-15
second10	-16
second5	-17
second	-18

Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  DateTimeOptions(ymd;2)
  ChartData(2006-08-15 2006-08-22 2006-08-27&12:00;
            19 28 23)
  AreaChart2D(;1)
  Scaling(x;linear;;;week)
  AxisMajorTickLabelTexts(x;"Week #|WY|")
CloseDrawing()

```

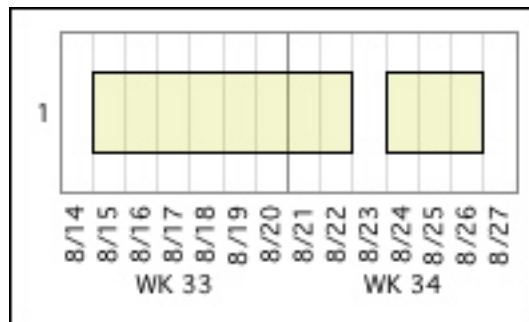




```

OpenDrawing(200;120;;2)
  AddFrame(0;0;200;120)
  DateTimeOptions(ymd;2)
  ChartData(2006-08-15 2006-08-22
            2006-08-24 2006-08-26)
  Ganttchart()
  FillStyle(1;200 200 0 50)
  Scaling(x;linear;;;week;day)
  AxisLine(all;0)
  AxisMajorTicks(all;0)
  AxisMinorTicks(all;0)
  AxisMajorTickLabelTexts(x;"WK |WY|")
  AxisMajorTickLabelOptions(x;out;;25)
  AxisMinorTickLabelTexts(x;"|M/D|")
  AxisMinorTickLabelStyle(x;;;;;-90)
CloseDrawing()

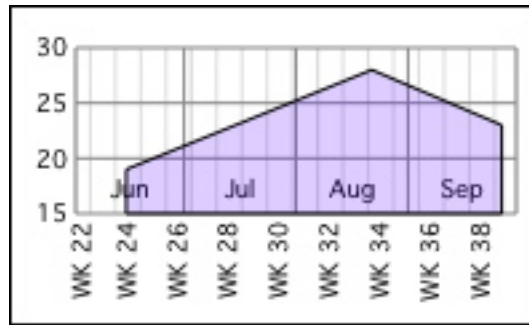
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  DateTimeOptions(ymd;2)
  ChartData(2006-06-15 2006-8-22 2006-9-27;
            19 28 23)
  AreaChart2D(;1)
  FillStyle(1;50 0 255 50)
  Scaling(x;linear;;;month;week)
  AxisLine(all;0)
  AxisMajorTicks(all;0)
  AxisMinorTicks(all;0)
  AxisMajorTickLabelTexts(x;"|Mon|")
  AxisMajorTickLabelOptions(x;in)
  AxisMinorTickLabelTexts(x;"WK |WY|")
  AxisMinorTickLabelStyle(x;;;;;-90)
  AxisMinorTickLabelOptions(x;;;2;;off)
CloseDrawing()

```

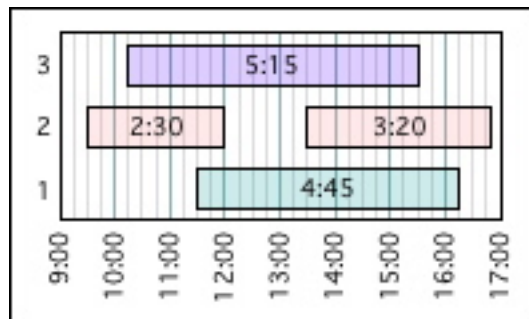


```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(11:30 16:15;
            9:30 12:00 13:30 16:50;
            10:15 15:30)
  GanttChart(label;55)
  FillStyle(1; 0 150 150 50)
  FillStyle(2;255 150 150 50)
  FillStyle(3; 50 0 255 50)

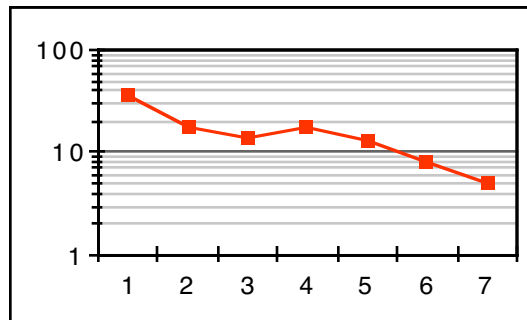
  Scaling(x;linear;;;hour;minute15)
  AxisLine(all;0)
  AxisMajorTicks(all;0)
  AxisMinorTicks(all;0)
  AxisMajorTickLabelTexts(x;"|h:mm|")
  AxisMajorTickLabelStyle(x;;;;;-90)
  MajorGridLineWidths(x;y;0)
  MajorGridLineColors(y;x;150 180 180)
  GridFrame()
CloseDrawing()

```

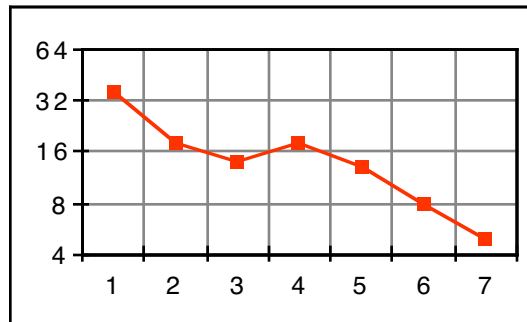


The arguments *logBaseValue* and *useEquidistantLogScaling* are solely used for logarithmic scaling. The base of the logarithm is defined by using the argument *logBaseValue*. If no base is defined, the decimal logarithm (*logBaseValue=10*) is used for scaling. By activating the argument *useEquidistantLogScaling=on*, only scaling values, which are a multiple of the base, are used; for example, for a logarithmic scaling with a base of 10, only values such as 0.01, 0.1, 1, 10, 100, 1000, etc. are used. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(36 18 14 18 13 8 5)
  LineChart(symbol;on)
  Scaling(y;log;;;9;10;on) // 9 minor intervals
  SymbolStyle(1;square;4;;red)
  MajorGridLineWidths(y;x;0) // hide vertical grid lines
  MajorGridLineColors;;;100 100 100)
  MinorGridLineColors;;;200 200 200)
  GridFrame()
CloseDrawing()
```



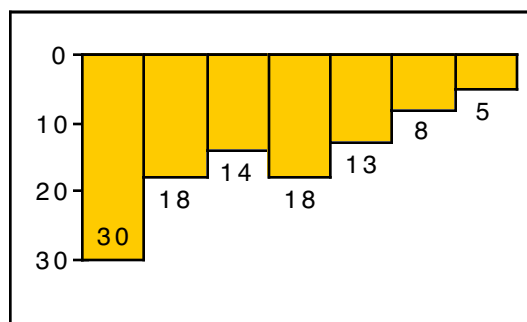
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(36 18 14 18 13 8 5)
  LineChart(symbol;on)
  Scaling(y;log;;;2;on) // logarithmic base = 2
  SymbolStyle(1;square;4;;red)
  GridFrame()
CloseDrawing()
```



**ScalingOptions(axisIndex;doReverseScaling;  
useIntegersOnly;hideZeroLabel)**

By activating the 2nd argument *doReverseScaling=on*, the direction of the scaling can be reversed, i.e. the scaling of the x-axis no longer increases from left to right, but the other way around, from right to left; the y-axis correspondingly increases from top to bottom. Examples:

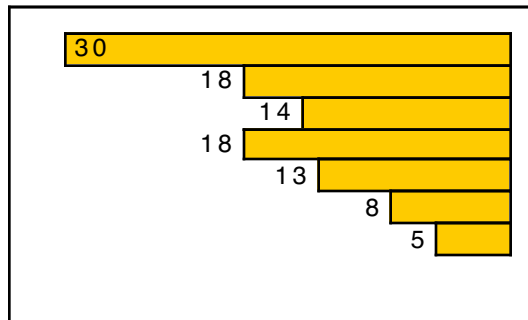
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(30 18 14 18 13 8 5)
  BarChart(label;0)
  FillStyle(1;darkYellow)
  ScalingOptions(y;on) // y-scaling top to bottom
  AxisOptions(x;none) // hide x-axis
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(30 18 14 18 13 8 5)
  BarChart(horizontal+label;0)
  FillStyle(1;darkYellow)
  ScalingOptions(all;on) // reverse scalings
  AxisOptions(all;none) // hide axes
  GridLocation(all;none) // hide grid
CloseDrawing()

```

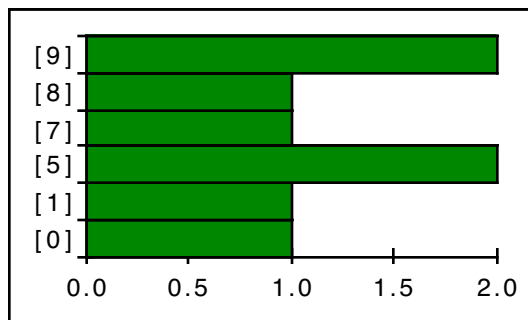


By activating the 3rd argument *useIntegersOnly=on*, scaling values which are not whole numbers can be suppressed. This proves useful when depicting integral values, e.g. for frequency distributions. Example:

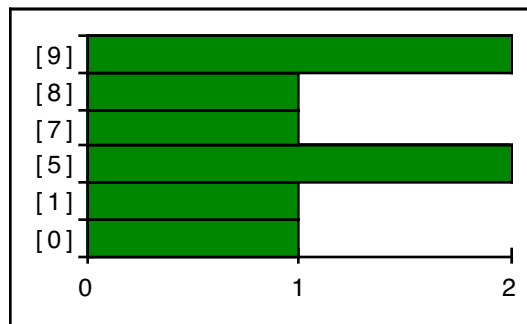
```

// without ScalingOptions()
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 9 0 9 5 5 7 8)
  Histogram(horizontal)
  HistogramOptions(on) // after Histogram()!
  FillStyle(all;green)
  GridLocation(xy;none) // hide grid
CloseDrawing()

```

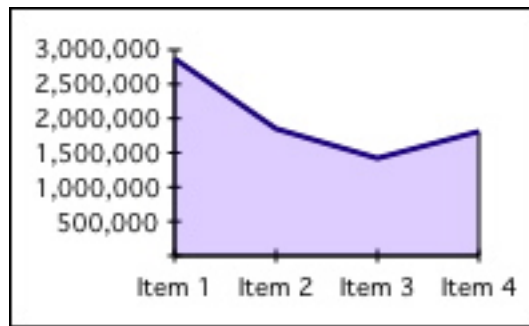


```
// with ScalingOptions()
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 9 0 9 5 5 7 8)
  Histogram(horizontal)
  HistogramOptions(on) // after Histogram()!
  FillStyle(all;green)
  GridLocation(xy;none) // hide grid
  ScalingOptions(x;;on) // integers only
CloseDrawing()
```



If the 4th argument *hideZeroLabel=on* is set, the scaling label for the value 0 is hidden. Sometimes this has proven to be advantageous when the scaling labels are very close to each other at the point where the x and y-axis intersect. Example:

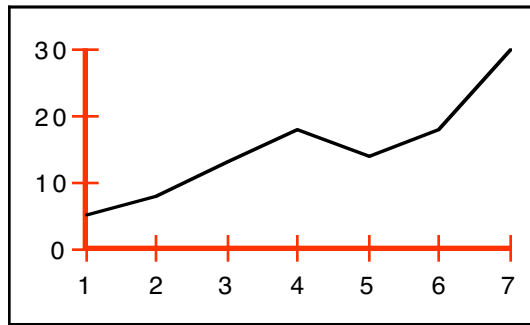
```
OpenDrawing(200;120)
  SetThousandsSep(",")
  AddFrame(0;0;200;120)
  ChartData(2870000 1850000 1420000 1810000)
  AreaChart()
  AreaChartOptions(on)
  FillStyle(1;50 0 255 50)
  LineStyle(1;;2;darkBlue)
  Scaling(y;linear;0;3000000;6)
  ScalingOptions(y;;;on) // hide "0"
  AxisMajorTickLabelTexts(x;"Item |u|")
  GridLocation(all;none) // hide grid
CloseDrawing()
```



**AxisLine(axisIndex;width;color;pattern)**  
**AxisMajorTicks(axisIndex;length;width;color;pattern;location)**  
**AxisMinorTicks(axisIndex;length;width;color;pattern;location)**

By using the `AxisLine()` function, the appearance of the axis lines can be controlled; by using the `AxisMajorTicks()` and `AxisMinorTicks()` functions, the appearance of the major and minor tick marks. As the default, the axis lines and tick marks are black and 1-pixel wide. The line width, color and pattern can be varied by the arguments *width*, *color* and *pattern*. In addition, using the arguments *length* and *location* the length (in pixels) and alignment of the tick marks, can be controlled. Tick marks can either be positioned inside the chart (*location=in*), outside the chart (*location=out*) or half inside and half outside the chart (*location=center*). The default major tick marks are five pixels long and the minor ones three pixels long. Examples:

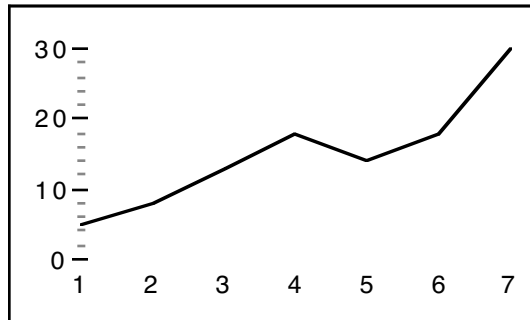
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 8 13 18 14 18 30)
  LineChart()
  LineStyle(1;poly;1;black)
  AxisLine(all;2;red)
  AxisMajorTicks(all;9;1;red)
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 8 13 18 14 18 30)
  LineChart()
  // 3 major and 5 minor intervals
  Scaling(y;linear;0;30;3;5)
  LineStyle(1;poly;1;black)
  AxisLine(all;0)           // hide axis lines
  AxisMajorTicks(x;0)       // hide x-axis tick marks
  AxisMajorTicks(y;5)
  AxisMinorTicks(y;2;1;gray)
  GridLocation(all;none) // hide grid
CloseDrawing()

```



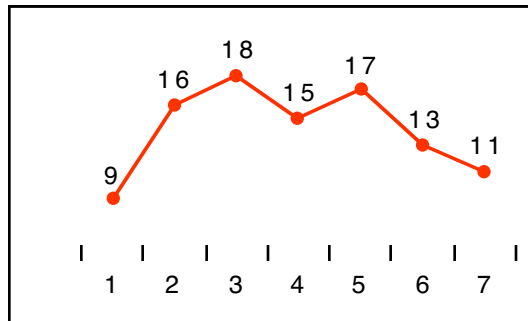


**AxisOptions(axisIndex;axisLocation;doShiftAxis)**

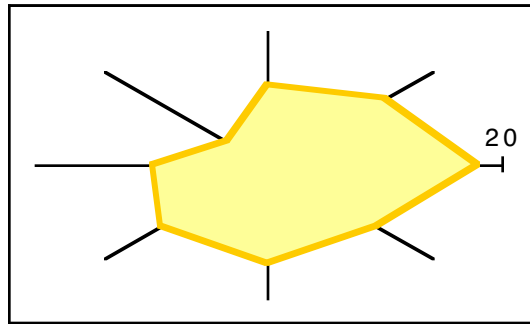
By using the 2nd argument *axisLocation*, axes can either be arranged behind the chart (*axisLocation=back*) or in front of it (*axisLocation=front*). The latter arrangement proves useful, e.g. for polar and radar charts. As an option, axes can be completely suppressed by *axisLocation=none*.

Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(9 16 18 15 17 13 11)
  LineChart(symbol+label;on)
  SymbolStyle(1;bullet;4;;red)
  LabelOptions(all;topCenter)
  AxisOptions(y;none)      // hide y-axis
  AxisLine(x;0)            // hide x-axis line
  GridLocation(all;none)  // hide grid
CloseDrawing()
```



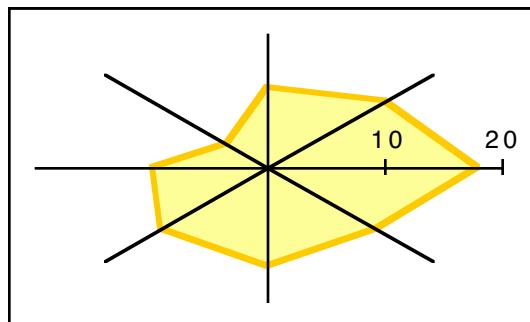
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(18 13 15 13 10 5 12 14)
  RadarChart(oval;90)
  FillStyle(1;lightYellow)
  BorderStyle(1;poly;2;darkYellow)
  AxisOptions(all;back)  // axes behind graph
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(18 13 15 13 10 5 12 14)
  RadarChart(oval;90)
  FillStyle(1;lightYellow)
  BorderStyle(1;poly;2;darkYellow)
  AxisOptions(all;front) // axes in front of graph
  GridLocation(all;none) // hide grid
CloseDrawing()

```

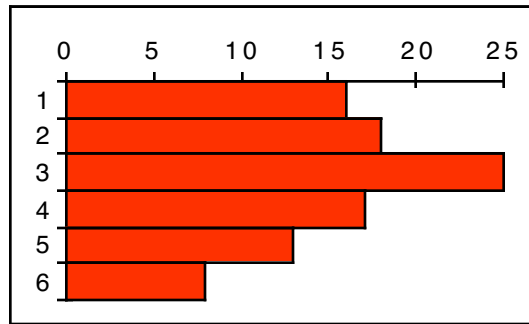


By activating the 3rd argument *doShiftAxis=on*, axes can be moved to the opposite side, i.e. the x-axis is moved from bottom to top, the y-axis from left to right. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(16 18 25 17 13 8)
  BarChart(horizontal;0)
  ScalingOptions(y;on) // y-scaling top to bottom
  AxisOptions(x;;on) // move x-axis
  GridLocation(all;none) // hide grid
CloseDrawing()

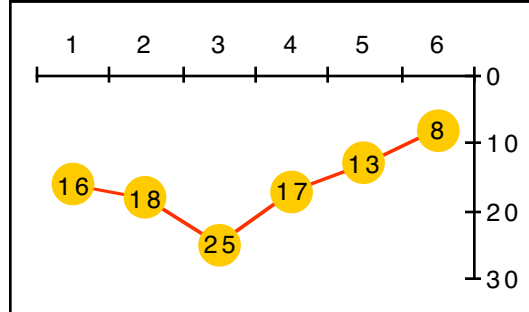
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(16 18 25 17 13 8)
  LineChart(symbol+label;on)
  ScalingOptions(y;on) // y-scaling top to bottom
  SymbolStyle(1;bullet;15;;darkYellow)
  LabelOptions(all;centerCenter)
  AxisOptions(all;;on) // move axes
  GridLocation(all;none) // hide grid
CloseDrawing()

```



### **AxisLabelText(axisIndex;text1;text2...)**

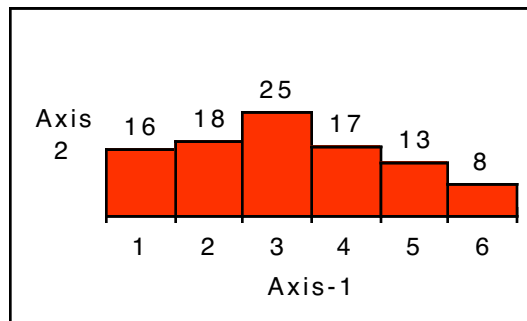
By using the `AxisLabelText()` function, axes can be given a label. Texts consisting of several lines are possible by inserting a line feed "`\n`". If the text contains a format specifier, e.g. "`|u|`", the axis index is displayed, i.e. "1" for the x-axis, "2" for the y-axis. The handling of texts in connection with format specifiers is discussed in combination with the `LabelTexts()` function in the *Styles* section.

Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(16 18 25 17 13 8)
  BarChart(label;0)
  AxisLine(y;0) // hide y-axis line
  AxisMajorTicks(y;0) // hide y-tick marks
  AxisMajorTickLabelTexts(y;"") // hide y-scaling labels
  AxisLabelText(x;"Axis-|u|")
  AxisLabelText(y;"Axis\n|u|")
  GridLocation(all;none) // hide grid
CloseDrawing()

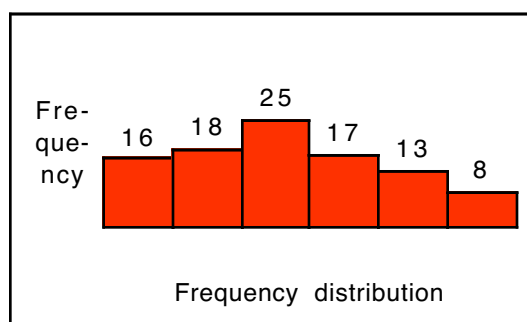
```



```

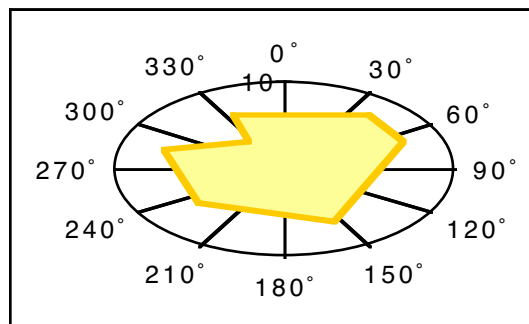
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(16 18 25 17 13 8)
  BarChart(label;0)
  AxisLine(all;0) // hide axis lines
  AxisMajorTicks(all;0) // hide tick marks
  AxisMajorTickLabelTexts(all;"") // hide scaling labels
  AxisLabelText(x;"Frequency distribution")
  AxisLabelText(y;"Fre-\nque-\nnncy")
  GridLocation(all;none) // hide grid
CloseDrawing()

```

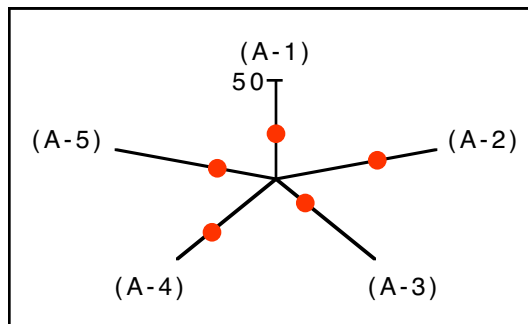


For polar and radar charts multiple texts can be entered which are appropriately assigned to the individual axes. If the number of texts entered is less than the number of axes, then the texts will be repeated periodically. If a text contains a format specifier, then the angle will be written in degrees on polar charts and the axis index on radar charts. Please note that on polar and radar charts *axisIndex=1* or *axisIndex=x* is to be set, all other axis indices are ignored. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(5 7 3 -5 -7 -2 -3; // x-values
            6 3 -6 -4 2 3 6) // y-values
  PolarChart(oval)
  FillStyle(1;lightYellow)
  BorderStyle(1;poly;2;darkYellow)
  MajorGridLineColors(all;all;black)
  AxisLabelText(all;"|u|°")
CloseDrawing()
```



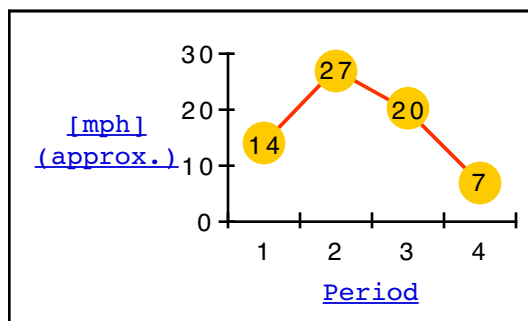
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(23 32 15 33 18)
  RadarChart(oval+symbol)
  FillStyle(;;transparent)
  BorderStyle(;;none)
  SymbolStyle(1;bullet;6;1;red)
  AxisLabelText(1;"(A-|u|)")
  GridLocation(all;none) // hide grid
CloseDrawing()
```



**AxisLabelStyle(axisIndex;font;size;style;color;  
alignment;orientation;maxWidth;  
maxHeight;ellipsisPosition)**

By using the AxisLabelStyle() function, it is possible to define a text style for axis labels. The attributes *orientation*, *maxWidth*, *maxHeight* and *ellipsisPosition* are discussed in the *Styles* section, LabelStyle() function. Examples:

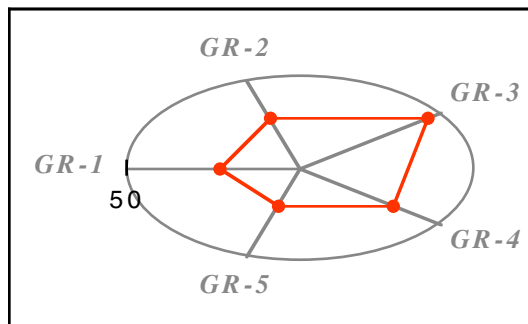
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(14 27 20 7)
  LineChart(symbol+label;on)
  SymbolStyle(1;bullet;15;;darkYellow)
  LabelOptions(all;centerCenter)
  AxisLabelText(x;"Period")
  AxisLabelText(y;"[mph]\n(approx.)")
  AxisLabelStyle(all;"Courier";10;underline;blue)
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(23 29 45 33 21)
  RadarChart(oval+symbol;-90)
  FillStyle(all;;;transparent)
  BorderStyle(1;poly;1;red)
  SymbolStyle(1;bulet;4;1;red)
  AxisLine(1;0)
  AxisLabelText(1;"GR-|u|")
  AxisLabelStyle(1;"Times";10;bold+italic;gray)
CloseDrawing()

```



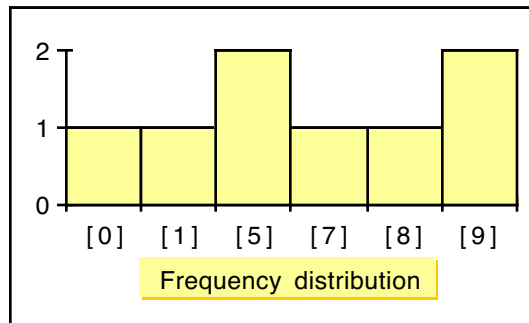
**AxisLabelBackground(axisIndex;fillColor;fillPattern;  
borderWidth;borderColor;borderPattern;  
shadowOffset;shadowColor;shadowPattern)**

By using the AxisLabelBackground() function, the background of the axis label can be designed. In doing so, it is possible to vary the color, pattern, border and shadow. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 9 0 9 5 5 7 8)
  Histogram()
  HistogramOptions(on) // after Histogram()!
  FillStyle(all;lightYellow)
  ScalingOptions(y;;on) // integers only
  AxisLabelText(x;" Frequency distribution ")
  AxisLabelBackground(x;lightYellow;;0;;1;darkYellow)
  GridLocation(xy;none) // hide grid
CloseDrawing()

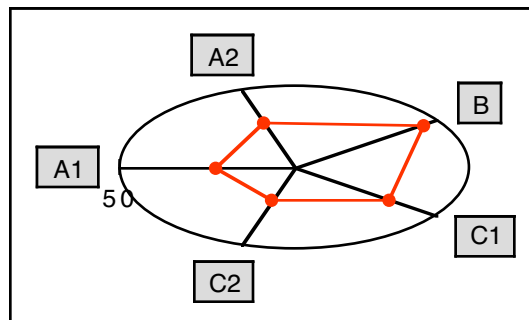
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(23 29 45 33 21)
  RadarChart(oval+symbol;-90)
  FillStyle(all;;transparent)
  BorderStyle(1;poly;1;red)
  SymbolStyle(1;bullet;4;1;red)
  AxisLine(1;0)
  AxisLabelText(1;" A1 "; " A2 "; " B "; " C1 "; " C2 ")
  AxisLabelBackground(x;lightGray;;1)
  MajorGridLineColors(all;all;black)
CloseDrawing()

```





**AxisLabelOptions(axisIndex;location;hOffset;vOffset)**

By using the AxisLabelOptions() function, the position of the axis label can be controlled. The argument *location* makes it possible to choose from among nine predefined positions:

<i>constant</i>	<i>value</i>
topLeft	1
topCenter	2
topRight	3
centerLeft	4
centerCenter	5
centerRight	6
bottomLeft	7
bottomCenter	8
bottomRight	9

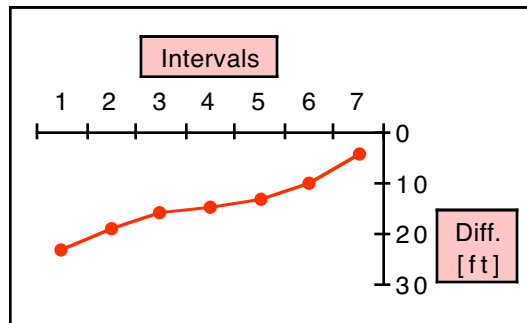
As the default, the x-axis is labeled in the center below the axis line (*location=bottomCenter*), the y-axis is labeled in the center left of the axis line (*location=centerLeft*). On polar and radar charts the axis labels are always located outside the chart; the argument *location* is ignored.

Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(23 19 16 15 13 10 4)
  LineChart(symbol;on)
  SymbolStyle(1;bullet;4)
  ScalingOptions(y;on) // y-scaling top to bottom
  AxisOptions(all;;on) // shift axes
  AxisLabelText(x;" Intervals ")
  AxisLabelText(y;" Diff. \n[ft]")
  AxisLabelBackground(all;255 200 200)
  AxisLabelOptions(x;topCenter)
  AxisLabelOptions(y;bottomRight)
  GridLocation(all;none) // hide grid
CloseDrawing()

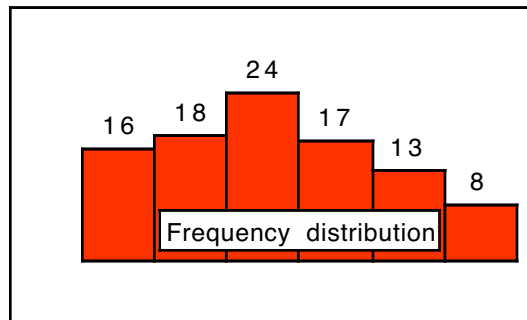
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(16 18 24 17 13 8)
  BarChart(label;0)
  AxisMajorTicks(x;0) // hide tick marks
  AxisMajorTickLabelTexts(x;"") // hide x-scaling labels
  AxisOptions(y;none)
  AxisLabelText(x;"Frequency distribution")
  AxisLabelBackground(x)
  AxisLabelOptions(x;topCenter)
  AxisOptions(x;front)
  GridLocation(all;none) // hide grid
CloseDrawing()

```



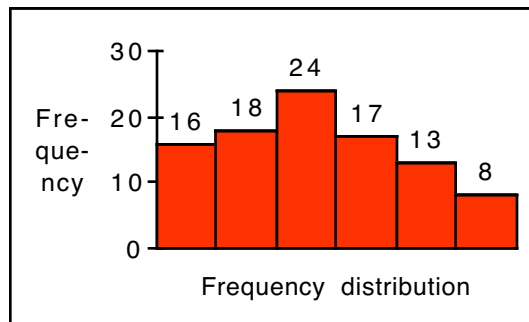
The arguments *hOffset* and *vOffset* can be used to finely adjust the position of the label. Positive offset values move the label down to the right, negative values up to the left.

Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(16 18 24 17 13 8)
  BarChart(label;0)
  AxisLabelText(x;"Frequency distribution")
  AxisLabelText(y;"Fre-\nque-\nnncy")
  AxisMajorTicks(x;0)           // hide tick marks
  AxisMajorTickLabelTexts(x;"") // hide x-scaling labels
  AxisLabelOptions(x;;;-10)     // move up 10 pixels
  AxisLabelOptions(y;;;-6)      // move left 6 pixels
  GridLocation(all;none)        // hide grid
CloseDrawing()

```



### **AxisMajorTickLabelTexts(axisIndex;text1;text2...)**

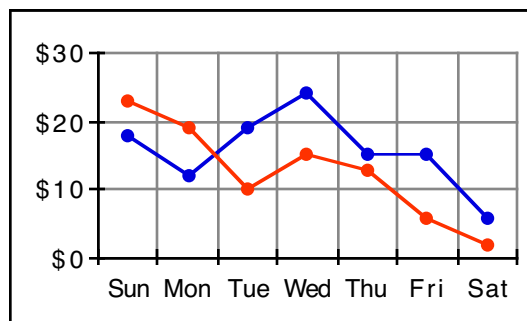
As the default, the scaling texts are the respective scaling values. By using the `AxisMajorTickLabelTexts()` function, the label text of the major tick marks can be controlled. Thus, it is possible to assign an individual text to each tick mark. The texts are repeated periodically if the number of major tick marks is larger than the number of texts entered. If the text contains a format specifier, e.g. "`|f1|`", the scaling value is accordingly formatted. The handling of texts in connection with format specifiers is discussed in the *Styles* section, function `LabelTexts()`. Texts consisting of several lines are possible by entering a line feed "`\n`".

Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(23 19 10 15 13 6 2;
            18 12 19 24 15 15 6)
  LineChart(symbol;on)
  SymbolStyle(1;bullet;4)
  SymbolStyle(2;bullet;4)
  AxisMajorTickLabelTexts(y;"$|u|")
  AxisMajorTickLabelTexts(x;"Sun";"Mon";"Tue";"Wed";
                            "Thu";"Fri";"Sat")
CloseDrawing()

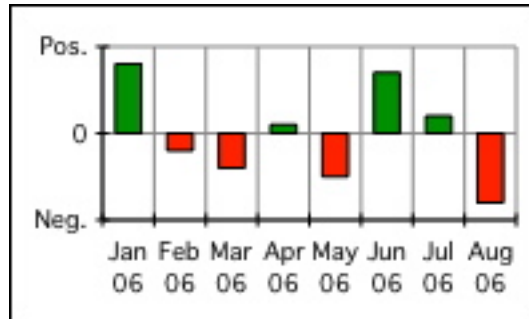
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(8 -2 -4 1 -5 7 2 -8)
  BarChart()
  BarChartOptions(;;on)
  Scaling(y;linear;-10;10;2)
  FillStyle(1;green)
  FillStyle(2;red)
  AxisMajorTickLabelTexts(y;"Neg."; "0"; "Pos.")
  AxisMajorTickLabelTexts(x;"Jan\n06"; "Feb\n06";
                            "Mar\n06"; "Apr\n06";
                            "May\n06"; "Jun\n06";
                            "Jul\n06"; "Aug\n06")
CloseDrawing()

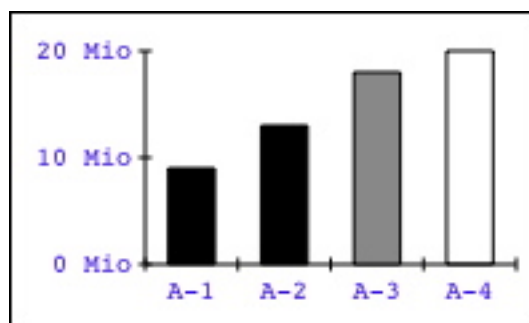
```



**AxisMajorTickLabelStyle(axisIndex;font;size;style;  
color;alignment;orientation;  
maxWidth;maxHeight;ellipsisPosition)**

The text style of the scaling labels can be varied by using the `AxisMajorTickLabelStyle()` function. The attributes *orientation*, *maxWidth*, *maxHeight* and *ellipsisPosition* are discussed in the *Styles* section, `LabelStyle()` function. Example:

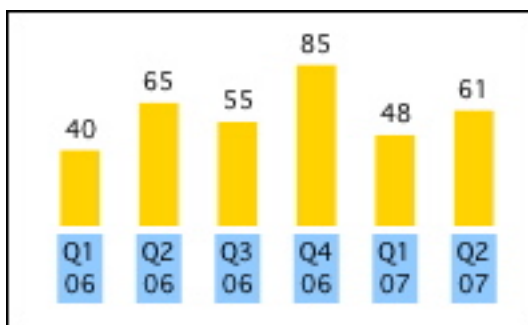
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 9000000 0 0 0;
             0 13000000 0 0;
             0 0 18000000 0;
             0 0 0 20000000)
  BarChart(stacked)
  FillStyle(all;black)
  FillStyle(3;gray)
  FillStyle(4;;transparent)
  AxisMajorTickLabelTexts(x;"A-1";"A-2";
                          "A-3";"A-4")
  AxisMajorTickLabelTexts(y;"|-6u| Mio")
  AxisMajorTickLabelStyle(all;"Courier";10;plain;blue)
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```
AxisMajorTickLabelBackground(axisIndex;fillColor;  
                             fillPattern;borderWidth;borderColor;  
                             borderPattern;shadowOffset;  
                             shadowColor;shadowPattern)
```

A background can be added to the tick mark labels by using the `AxisMajorTickLabelBackground()` function. It is possible to vary the color, pattern, border and shadow. Examples:

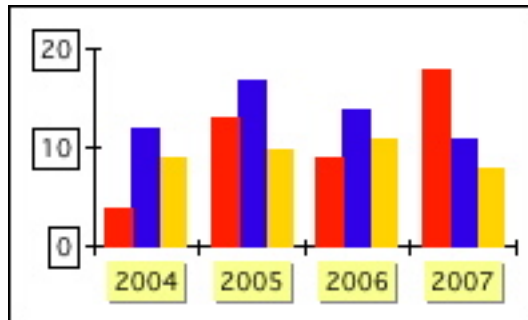
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(40 65 55 85 48 61)
  BarChart(label)
  FillStyle(1;darkYellow)
  BorderStyle(1;none)
  LabelOptions(1;out) // place labels outside of bars
  AxisLine(all;0) // hide axis line
  AxisMajorTicks(all;0) // hide tick marks
  AxisMajorTickLabelTexts(y;"") //hide y-scaling labels
  AxisMajorTickLabelTexts(x;"Q1\n06";"Q2\n06";  
                           "Q3\n06";"Q4\n06";  
                           "Q1\n07";"Q2\n07")
  AxisMajorTickLabelBackground(x;lightBlue;;0)
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 13 9 18;
            12 17 14 11;
            9 10 11 8)
  BarChart()
  FillStyle(3;darkYellow)
  BorderStyle(all;none)
  AxisMajorTickLabelTexts(x;"2004";"2005";
                          "2006";"2007")
  AxisMajorTickLabelBackground(x;lightYellow;;0;;1)
  AxisMajorTickLabelBackground(y)
  GridLocation(all;none)
CloseDrawing()

```



**AxisMajorTickLabelOptions(axisIndex;location;hOffset  
vOffset;labelEveryNthTickMark;startAtTickMark)**

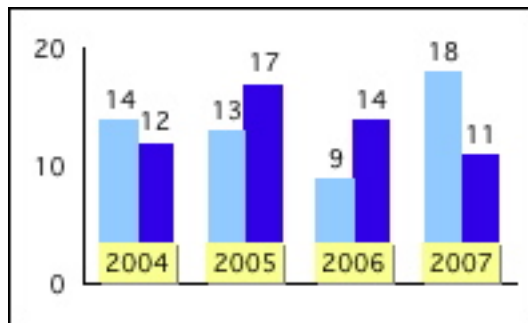
The `AxisMajorTickLabelOptions()` function serves to position and finely adjust the scaling labels. The argument *location* makes it possible to position the labels either inside the chart (*location=in*) or outside the chart (*location=out*) or directly on the axis line (*location=center*). By using the arguments *hOffset* and *vOffset*, the location of the labels can be finely adjusted. Positive offset values move the labels down to the right, negative values up to the left.

## Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(14 13 9 18; 12 17 14 11)
  BarChart(label)
  FillStyle(1;lightBlue)
  BorderStyle(all;none)
  LabelOptions(all;out) // place labels outside of bars
  AxisMajorTicks(all;0) // hide tick marks
  AxisMajorTickLabelTexts(x;"2004";"2005";
                          "2006";"2007")
  AxisMajorTickLabelBackground(x;lightYellow;;0;;1)
  AxisMajorTickLabelOptions(x;in;;2) // 2 pixels down
  AxisMajorTickLabelOptions(y;;-3)   // 3 pixels left
  AxisOptions(x;front) // axis labels in front of chart
  GridLocation(all;none)
CloseDrawing()

```

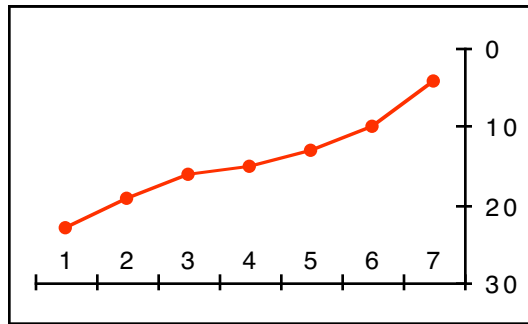


```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(23 19 16 15 13 10 4)
  LineChart(symbol;on)
  SymbolStyle(1;bullet;4)
  ScalingOptions(y;on) // y-scaling top to bottom
  AxisOptions(y;on)    // move y-axis to the right
  AxisMajorTickLabelOptions(x;in;;3) // 3 pixels down
  AxisMajorTickLabelOptions(y;;3)    // 3 pixels right
  GridLocation(all;none)
CloseDrawing()

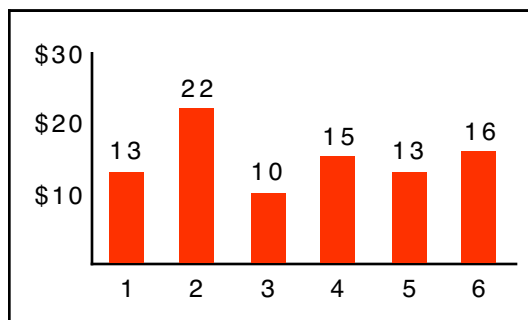
```





Scaling labels can be suppressed by using the arguments *labelEveryNthTickMark* and *startAtTickMark*. For example, when the 3rd tick mark, 5th tick mark, 7th tick mark, etc. are to be labeled, the arguments *labelEveryNthTickMark* and *startAtTickMark* should be set to 2 and 3, respectively. Examples:

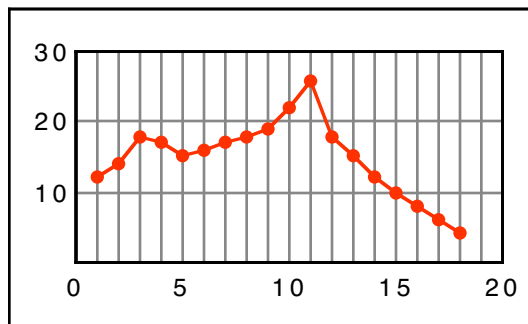
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(13 22 10 15 13 16)
  BarChart(label)
  BorderStyle(all;none)
  AxisMajorTicks(all;0) // hide tick marks
  AxisMajorTickLabelTexts(y;"$|u|")
  AxisMajorTickLabelOptions(y;;;1;2) // hide "$0"
  GridLocation(all;none)
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(12 14 18 17 15 16 17 18 19
            22 26 18 15 12 10 8 6 4)
  LineChart(symbol)
  SymbolStyle(all;bullet;4)
  Scaling(x;linear;0;20;20)
  AxisMajorTicks(all;0)
  AxisMajorTickLabelOptions(x;;;5) // every 5th value
  AxisMajorTickLabelOptions(y;;;1;2) // hide "0"
  GridFrame()
CloseDrawing()

```



```

AxisMinorTickLabelTexts(axisIndex;text1;text2...)
AxisMinorTickLabelStyle(axisIndex;font;size;style;
                        color;alignment;orientation;maxWidth;
                        maxHeight;ellipsisPosition)
AxisMinorTickLabelBackground(axisIndex;fillColor;
                             fillPattern;borderWidth;borderColor;
                             borderPattern;shadowOffset;
                             shadowColor;shadowPattern)
AxisMinorTickLabelOptions(axisIndex;location;hOffset;
                          vOffset;labelEveryNthTickMark;
                          startAtTickMark)

```

The functions for labeling and designing the minor tick marks work exactly the same way as the functions for labeling the major tick marks.

## Examples:

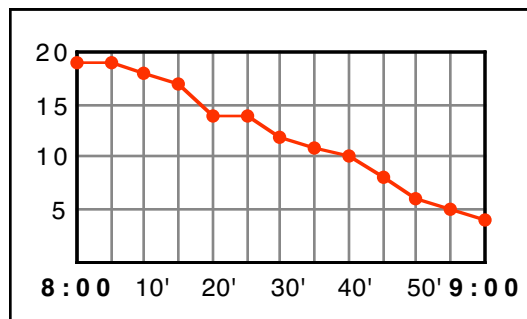
```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 1  2  3  4  5  6  7  8  9 10 11 12 13;
            19 19 18 17 14 14 12 11 10  8  6  5  4)
  LineChart2D(symbol)
  Scaling(x;linear;1;13;1;12)
  SymbolStyle(1;bullet;4)

  // axes
  AxisMajorTicks(all;0) // hide major tick marks
  AxisMinorTicks(all;0) // hide minor tick marks
  AxisMajorTickLabelTexts(x;"8:00";"9:00")
  AxisMajorTickLabelStyle(x;;;bold)
  AxisMinorTickLabelTexts(x;" 10'";" 20'";" 30'";
                          " 40'";" 50'")
  AxisMinorTickLabelOptions(x;;;2;2) // every 2nd value
  AxisMajorTickLabelOptions(y;;;1;2) // hide "0"

  // grid
  MajorGridLinePatterns(all;all;black)
  MajorGridLineColors(all;all;gray)
  MinorGridLinePatterns(all;all;black)
  MinorGridLineColors(all;all;gray)
  GridFrame()
CloseDrawing()

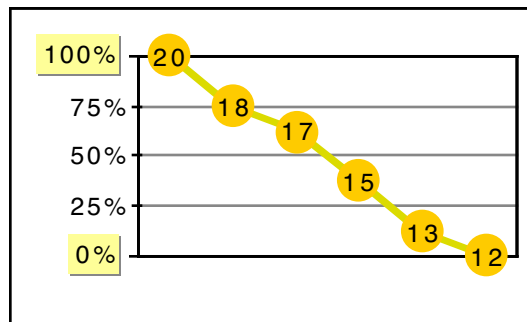
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(20 18 17 15 13 12)
  LineChart(symbol+label;on)
  Scaling(y;percent;0;100;1;4)
  LineStyle(1;poly;2;220 220 0)
  SymbolStyle(1;bullet;15;;darkYellow)
  LabelOptions(all;centerCenter)
  AxisOptions(x;none) // hide x-axis
  AxisMajorTickLabelBackground(y;lightYellow;;0;;1)
  AxisMinorTickLabelTexts(y;"|u|%" )
  MajorGridLineWidths(y;x;0) // hide vertical grid lines
  GridFrame()
CloseDrawing()

```



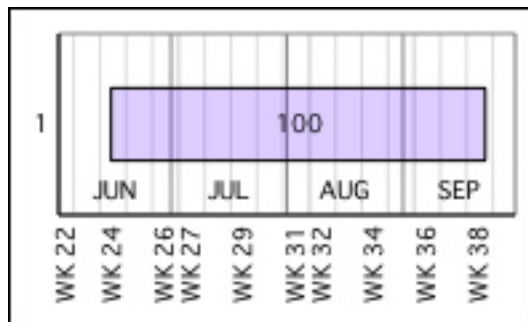
Unlike the function `AxisMajorTickLabelOptions()`, the function `AxisMinorTickLabelOptions()` has an additional argument *repeatLabelPattern*. As the default, the label pattern of the minor tick mark texts defined by the arguments *labelEveryNthTickMark* and *startAtTickMark* is reset at every major tick mark and starts again from the beginning (*repeatLabelPattern=on*). If *repeatLabelPattern=off* is set, then the label pattern is not repeated, but rather runs through the entire scaling regardless of the major tick marks. This proves to be advantageous, e.g. for weekly scalings, as the minor scaling (week) does not exactly match the major scaling (year, quarter or month).

Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  DateTimeOptions(ymd;2)
  ChartData(2006-6-15 2006-9-22)
  GanttChart(label;150)
  FillStyle(1;50 0 255 50)
  Scaling(x;linear;;;month;week)
  AxisMajorTicks(all;0)
  AxisMinorTicks(all;0)
  AxisMajorTickLabelTexts(x;"|MON|")
  AxisMajorTickLabelOptions(x;in)
  AxisMinorTickLabelTexts(x;"WK |WY|")
  AxisMinorTickLabelStyle(x;;;;;-90)
  AxisMinorTickLabelOptions(x;;;2;;on)
CloseDrawing()

```

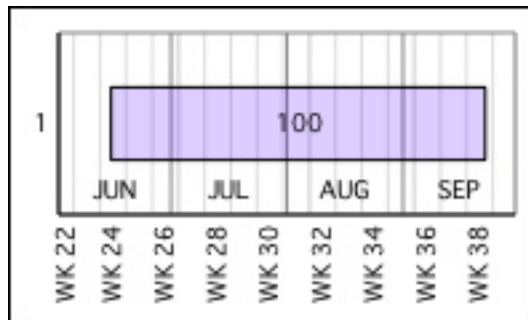


The same example with *repeatLabelPattern=off*:

```

...
AxisMinorTickLabelOptions(x;;;2;;off)

```



## Legend

Four functions are available for creating legends. They make it possible to:

- define the legend texts
- define the text style
- design the background
- position the legend in a flexible manner
- finely tune it using numerous options

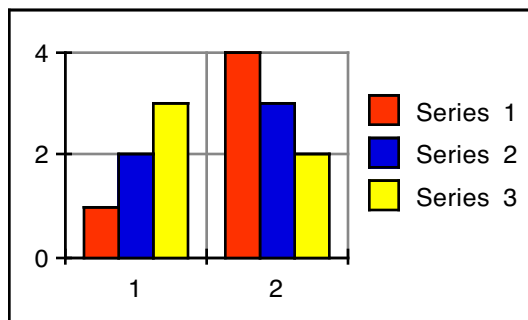
A legend is not created automatically by default, but rather by calling one of the four following functions.

### **LegendTexts(text1;text2...)**

By using the `LegendTexts()` function each data series can be assigned a name. Texts consisting of several lines are possible by inserting a line feed "\n". If no or too few names are provided, the default text "Series" plus a serial number will be used for the missing names.

Examples:

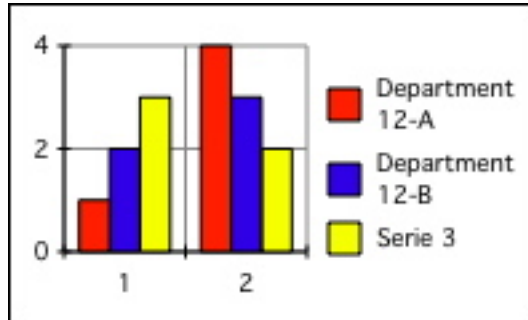
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 4; 2 3; 3 2)
  BarChart()
  LegendTexts()
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 4; 2 3; 3 2)
  BarChart()
  LegendTexts("Department\n12-A";
              "Department\n12-B")
CloseDrawing()

```



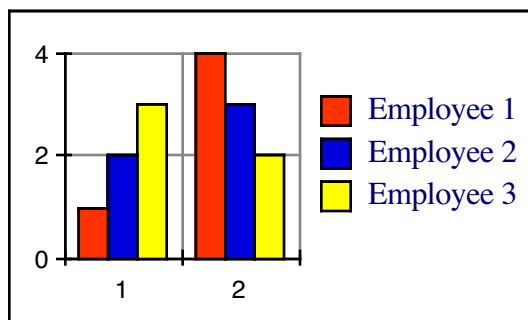
**LegendStyle(font;size;style;color;alignment;  
orientation;maxWidth;maxHeight;  
ellipsisPosition)**

The text style of the legend is defined by using the LegendStyle() function. The attributes *orientation*, *maxWidth*, *maxHeight* and *ellipsisPosition* are discussed in the *Styles* section, LabelStyle() function. For example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 4; 2 3; 3 2)
  BarChart()
  LegendTexts("Employee 1";
              "Employee 2";
              "Employee 3")
  LegendStyle("Times";12;plain;darkBlue)
CloseDrawing()

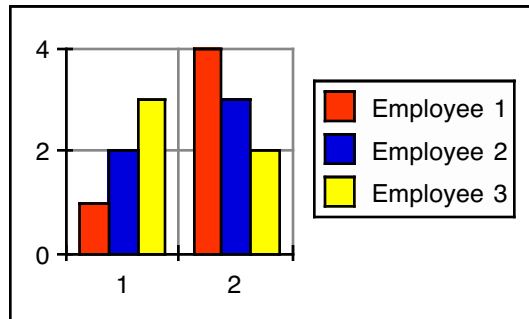
```



**LegendBackground(fillColor;fillPattern;borderWidth;  
borderColor;borderPattern;shadowOffset;  
shadowColor;shadowPattern)**

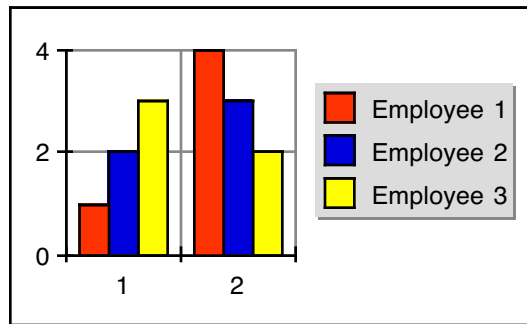
The background of the legend can be designed by using the LegendBackground() function. It is possible to vary the color, pattern, border and shadow as well. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 4; 2 3; 3 2)
  BarChart()
  LegendTexts("Employee 1";
              "Employee 2";
              "Employee 3")
  LegendBackground()
CloseDrawing()
```



```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 4; 2 3; 3 2)
  BarChart()
  LegendTexts("Employee 1";
              "Employee 2";
              "Employee 3")
  LegendBackground(lightGray;;0;;;1)
CloseDrawing()
```





**LegendOptions(location;placeInside;hOffset;vOffset;  
distribution;markerType;markerWidth;  
markerHeight;markerGap;rowGap;columnGap;  
textLocation;markerShape)**

The LegendOptions() function provides numerous options for positioning and finely tuning the legend. As the default, the legend is positioned on the right side of the chart. The 1st argument, *location*, makes it possible to choose from nine predefined positions:

<i>constant</i>	<i>value</i>
topLeft	1
topCenter	2
topRight	3
centerLeft	4
centerCenter	5
centerRight	6 (default)
bottomLeft	7
bottomCenter	8
bottomRight	9

Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 4; 2 3; 3 2)
  BarChart()
  LegendTexts("Employee 1";
              "Employee 2";
              "Employee 3")
  LegendOptions(centerLeft)
CloseDrawing()
```



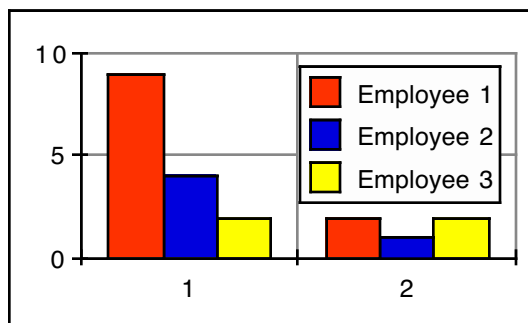
In addition, it is also possible to position the legend relative to the coordinate origin by using *location=0*. The coordinate origin is located in the upper left-hand corner of the drawing or, if there is a view, in the upper left-hand corner of the view.

The 2nd argument, *placeInside*, makes it possible to position the legend within the area of the chart. The location of the legend can also be finely adjusted by *hOffset* and *vOffset*. Positive offset values move the legend down to the right, negative values up to the left. Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(9 2; 4 1; 2 2)
  BarChart()
  LegendTexts("Employee 1";
              "Employee 2";
              "Employee 3")
  LegendBackground()
  LegendOptions(topRight;on;3;-3)
CloseDrawing()

```



The layout of the legend texts can be designed in a wide variety of ways using the argument *distribution*. The argument *distribution* can consist of up to 6 elements which are separated by spaces, tabs or line feeds.

Element	description	range	default
[1]	number of rows	-1..1000	10
[2]	number of columns	-1..1000	-1
[3]	arrange column by column	0..1	0
[4]	arrange reversed	0..1	0
[5]	equidistant column widths	0..1	0
[6]	equidistant row heights	0..1	0

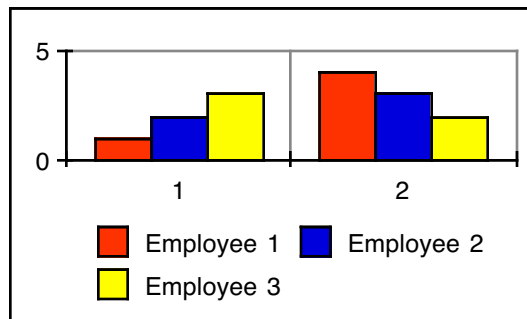
(-1...variable number of rows or columns)

Examples:

```
LegendOptions(;;;2)           // 2 rows (variable # of columns)
LegendOptions(;;;2 -1)        // 2 rows (same as above)
LegendOptions(;;;-1 2)        // 2 columns (variable # of rows)
LegendOptions(;;;3 -1 1)      // 3 rows (column by column)
LegendOptions(;;;-1 1 0 1)    // 1 column, reversed
```

Example:

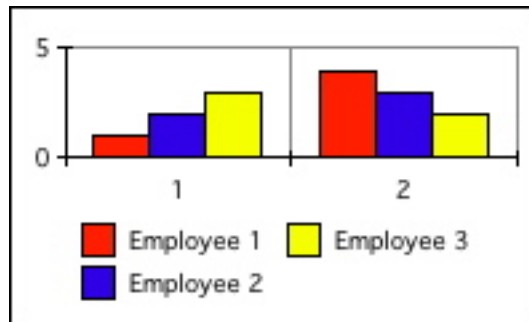
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 4; 2 3; 3 2)
  BarChart()
  LegendTexts("Employee 1";
              "Employee 2";
              "Employee 3")
  // arrange row by row
  LegendOptions(bottomCenter;;;2)
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 4; 2 3; 3 2)
  BarChart()
  LegendTexts("Employee 1";
              "Employee 2";
              "Employee 3")
  // arrange column by column
  LegendOptions(bottomCenter;;;2 -1 1)
CloseDrawing()

```



```

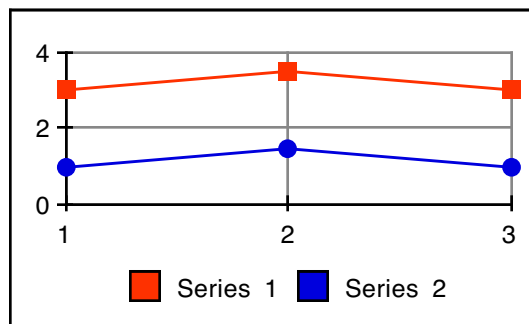
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1; 4; 3)
  BarChart(horizontal)
  LegendTexts("Employee 1";
              "Employee 2";
              "Employee 3")
  // arrange reversed, column by column
  LegendOptions(centerLeft;;;-8;
               -1 1 1 1;
               ;;;;12)
CloseDrawing()

```

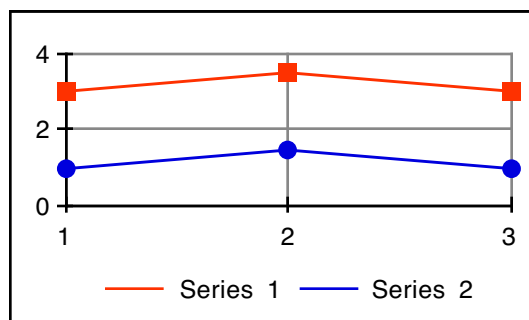


The argument, *markerType*, serves to define the legend symbols. It is also possible to combine rectangle, line and chart symbols simultaneously. If no marker type is defined, an appropriate type is automatically chosen, e.g. a square for bar and pie charts and a line combined with a symbol for line charts with symbols. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 3.5 3; 1 1.5 1)
  LineChart(symbol)
  SymbolStyle(1;square;6)
  SymbolStyle(2;bulet;6)
  LegendOptions(bottomCenter;;;1;rect)
CloseDrawing()
```



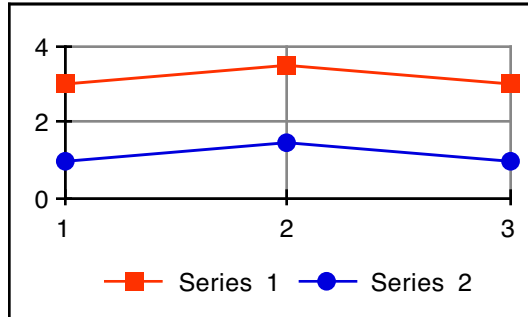
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 3.5 3; 1 1.5 1)
  LineChart(symbol)
  SymbolStyle(1;square;6)
  SymbolStyle(2;bulet;6)
  LegendOptions(bottomCenter;;;1;line)
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 3.5 3; 1 1.5 1)
  LineChart(symbol)
  SymbolStyle(1;square;6)
  SymbolStyle(2;bulet;6)
  LegendOptions(bottomCenter;;;1;line+symbol)
CloseDrawing()

```

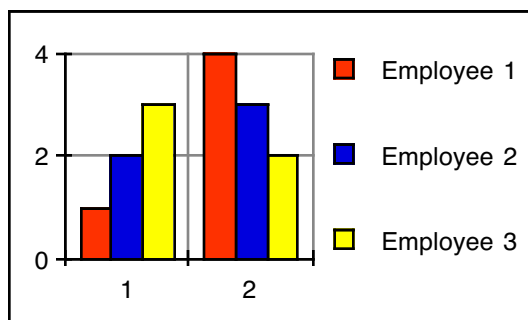


The arguments, *markerWidth*, *markerHeight*, *markerGap*, *rowGap* and *columnGap*, control the size of the legend symbol, the space between the legend symbol and legend text as well as the space between the legend texts. Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 4; 2 3; 3 2)
  BarChart()
  LegendTexts("Employee 1";
              "Employee 2";
              "Employee 3")
  LegendOptions(;;;;;8;8;10;20)
CloseDrawing()

```



By using the argument *textLocation*, the layout of the legend text can be controlled in relation to the legend symbol. Nine positions are available. As the default, legend texts are positioned to the right of the legend symbol (*textLocation=centerRight*).

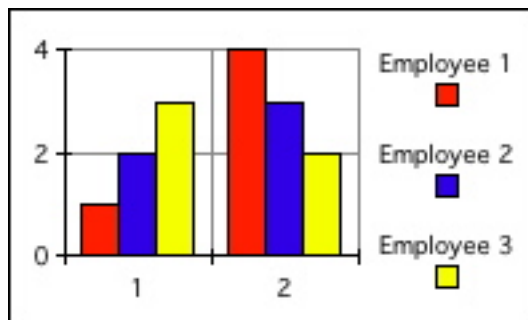
<i>constant</i>	<i>value</i>
topLeft	1
topCenter	2
topRight	3
centerLeft	4
centerCenter	5
centerRight	6 (default)
bottomLeft	7
bottomCenter	8
bottomRight	9

Examples:

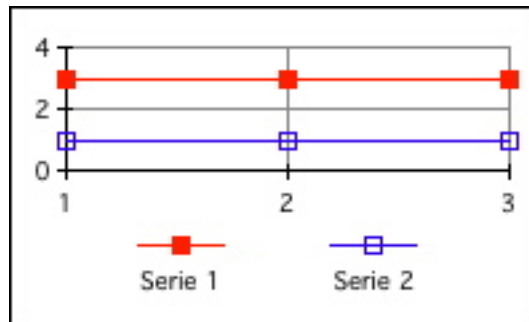
```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(1 4; 2 3; 3 2)
  BarChart()
  LegendTexts("Employee 1";
              "Employee 2";
              "Employee 3")
  LegendOptions(;;;;;
               8;8; // symbol size
               2;  // gap between symbol and text
               12;; // row gap
               topCenter)
CloseDrawing()

```



```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 3.0 3; 1 1.0 1)
  LineChart(symbol)
  SymbolStyle(1;square;6)
  SymbolStyle(2;hollowSquare;6)
  LegendOptions(bottomCenter;;;1;line+symbol)
  LegendOptions(bottomCenter;;;
    1;      // number of rows
    line+symbol;
    18;12; // symbol size
    4;     // gap between symbol and text
    0;40;  // column gap
    bottomCenter)
CloseDrawing()
```





# Title

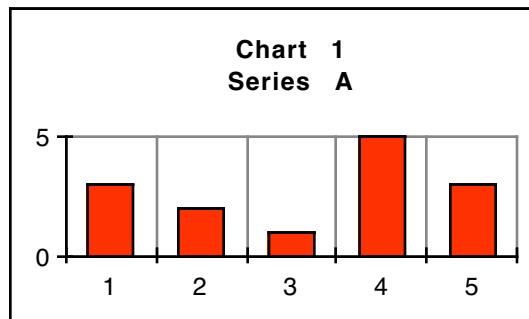
A total of five functions are available to set up the title. They make it possible to:

- define a title text incl. subtitle
- use separate text styles for the title and subtitle
- design the background
- position the title in a flexible manner

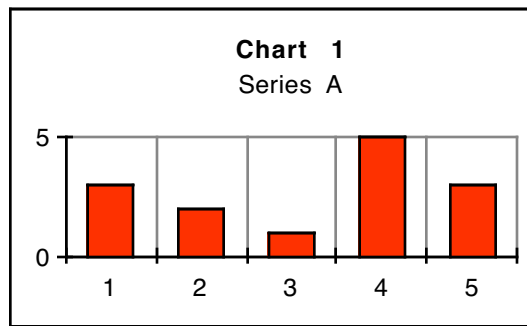
## **TitleText(title;subtitle)**

The TitleText() function is used to define the text for the title; it is also possible to define a subtitle as the 2nd argument. Texts consisting of several lines can also be entered by inserting a line feed "\n". Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 2 1 5 3)
  BarChart()
  TitleText("Chart 1\nSeries A")
CloseDrawing()
```



```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 2 1 5 3)
  BarChart()
  TitleText("Chart 1";"Series A")
CloseDrawing()
```

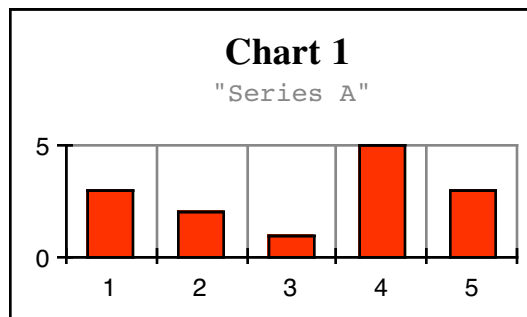


```
TitleStyle(font;size;style;color;alignment;  
orientation;maxWidth;maxHeight;  
ellipsisPosition)
```

```
TitleSubStyle(font;size;style;color;alignment;  
orientation;maxWidth;maxHeight;  
ellipsisPosition)
```

The `TitleStyle()` and `TitleSubStyle()` functions are used to define the text style for the title and subtitle. The attributes *orientation*, *maxWidth*, *maxHeight* and *ellipsisPosition* are discussed in the *Styles* section, `LabelStyle()` function. For example:

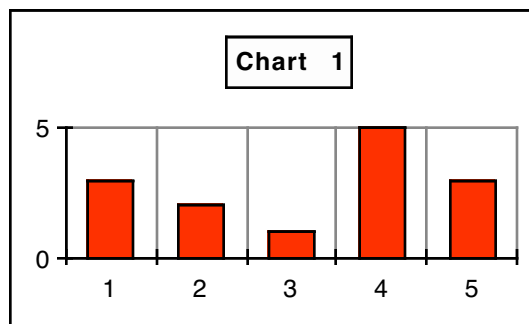
```
OpenDrawing(200;120)  
  AddFrame(0;0;200;120)  
  ChartData(3 2 1 5 3)  
  BarChart()  
  TitleText("Chart 1";"\\"Series A\\"")  
  TitleStyle("Times";14;bold)  
  TitleSubStyle("Courier";10;plain;gray)  
CloseDrawing()
```



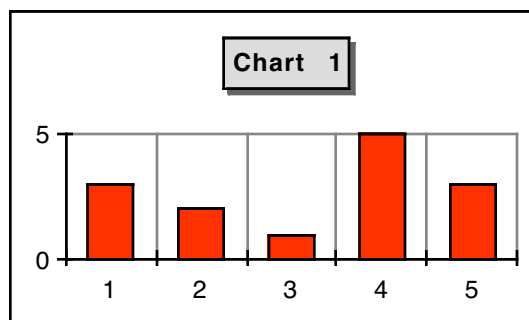
```
TitleBackground(fillColor;fillPattern;borderWidth;  
borderColor;borderPattern;shadowOffset;  
shadowColor;shadowPattern)
```

The `TitleBackground()` function can be used to design the background of the title. In doing so, it is possible to vary the color, pattern, border and shadow. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 2 1 5 3)
  BarChart()
  TitleText("Chart 1")
  // white background with a
  // 1 pixel wide black border.
  TitleBackground()
CloseDrawing()
```



```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 2 1 5 3)
  BarChart()
  TitleText("Chart 1")
  // light gray background with a 1 pixel
  // wide black border and a 2 pixels
  // wide dark gray shadow.
  TitleBackground(lightGray;;1;;2;darkGray)
CloseDrawing()
```



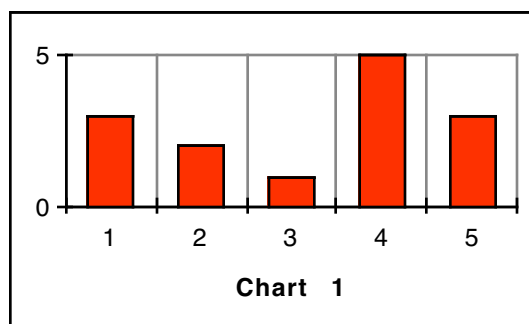
**TitleOptions(location;placeInside;hOffset;vOffset;  
vSubtitleOffset;titleAlignment)**

The TitleOptions() function is used to position and align the title. As the default, the title is positioned in the center at the top of the chart. The 1st argument, *location*, makes it possible to choose from among nine predefined positions:

<i>constant</i>	<i>value</i>
topLeft	1
topCenter	2 (default)
topRight	3
centerLeft	4
centerCenter	5
centerRight	6
bottomLeft	7
bottomCenter	8
bottomRight	9

Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 2 1 5 3)
  BarChart()
  TitleText("Chart 1")
  TitleOptions(bottomCenter)
CloseDrawing()
```



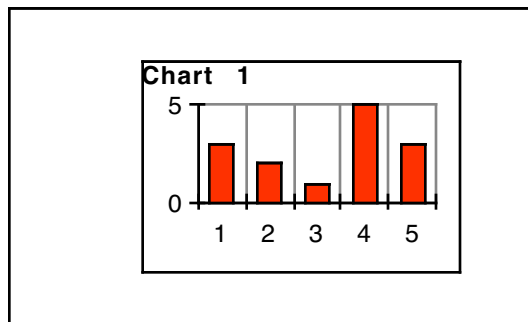
In addition, it is also possible to position the title relative to the coordinate origin by using *location=0*. The coordinate origin is located in the upper left-hand corner of the drawing or, if the chart is placed within a view, in the upper left-hand corner of the view.

Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  OpenView(50;20;120;80)
    AddFrame(0;0;120;80)
    ChartData(3 2 1 5 3)
    BarChart()
    TitleText("Chart 1")
    TitleOptions(0)
  CloseView()
CloseDrawing()

```

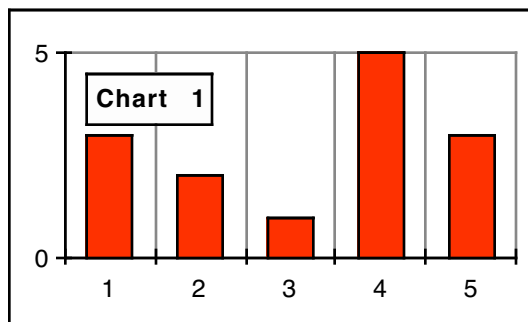


The 2nd argument, *placeInside*, makes it possible to position the title within the area of the chart. Example:

```

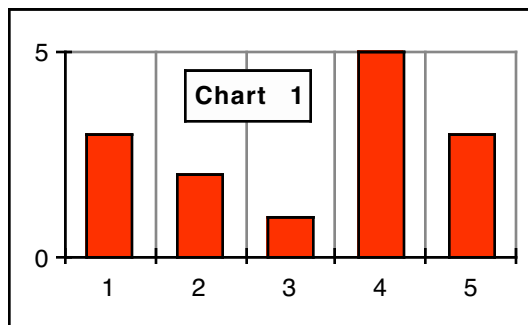
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 2 1 5 3)
  BarChart()
  TitleText("Chart 1")
  TitleBackground()
  TitleOptions(topLeft;on)
CloseDrawing()

```



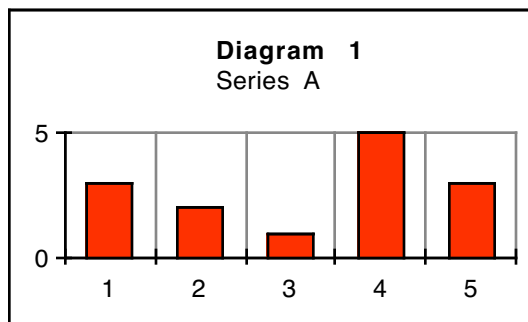
The location of the title can also be finely adjusted by *hOffset* and *vOffset*. Positive offset values move the title down to the right, negative values up to the left. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 2 1 5 3)
  BarChart()
  TitleText("Chart 1")
  TitleBackground()
  TitleOptions(topLeft;on;37;-1)
CloseDrawing()
```



The space and alignment between the title and subtitle can be controlled by *vSubtitleOffset* and *titleAlignment*. A positive offset value increases the space between the title and subtitle, a negative value decreases the space. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 2 1 5 3)
  BarChart()
  TitleText("Diagram 1";"Series A")
  TitleOptions(topCenter;off;0;0;-2;left)
CloseDrawing()
```



## Drop Lines

As an option, drop lines can be added to scatter, line and bubble charts. A total of four functions are available for this. They serve to:

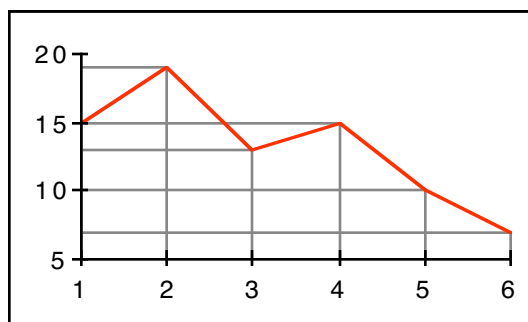
- design the drop lines
- define a reference point
- define a reference line
- define a set of reference series

For all four functions the series index is defined as the 1st argument. If no series index is defined or if *seriesIndex=all* is set, the respective function refers to all data series.

**DropLineStyle(seriesIndex;dropLineAxisIndex;width;  
color;pattern)**

The 2nd argument *dropLineAxisIndex* defines the direction of the drop lines. When *dropLineAxisIndex=x*, the drop lines refer to the x-axis; when *dropLineAxisIndex=y*, they refer to the y-axis. If no axis index is defined or if *dropLineAxisIndex=all*, drop lines are drawn in the direction of both the x-axis and y-axis. The line width, color and pattern of the drop lines can be varied using the arguments *width*, *color* and *pattern*. Unless otherwise defined, drop lines are gray, 1-pixel wide lines. Examples:

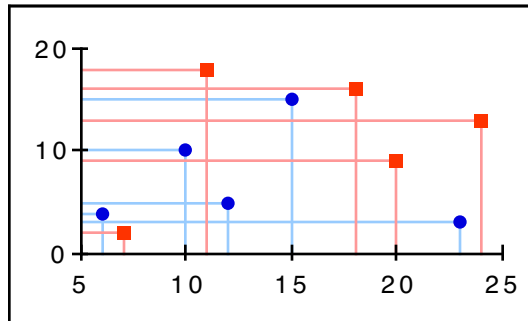
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(15 19 13 15 10 7)
  LineChart()
  DropLineStyle()
  AxisOptions(all;front) // axes in front of drop lines
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 7 11 18 20 24; // 1st series (x-values)
            2 18 16 9 13; // 1st series (y-values)
            23 10 15 12 6; // 2nd series (x-values)
            3 10 15 5 4) // 2nd series (y-values)
  ScatterChart2D()
  SymbolStyle(1;square;4;;red)
  SymbolStyle(2;bulet;4;;blue)
  DropLineStyle(1;all;1;lightRed)
  DropLineStyle(2;all;1;lightBlue)
  AxisOptions(all;front) // axes in front of drop lines
  GridLocation(all;none) // hide grid
CloseDrawing()

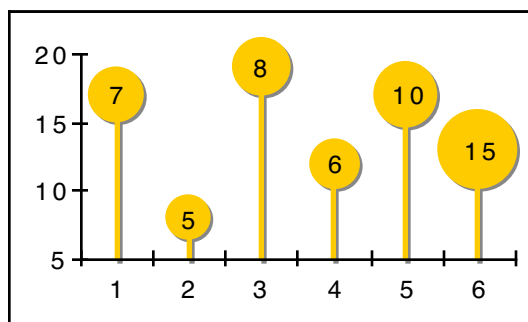
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(17 8 19 12 17 13; // y-values
            7 5 8 6 10 15) // diameters
  BubbleChart(label+shadow;on)
  FillStyle(1;darkYellow)
  BorderStyle(1;none)
  ShadowStyle(1;1;gray)
  DropLineStyle(1;x;2;darkYellow)
  AxisOptions(all;front) // axes in front of drop lines
  GridLocation(all;none) // hide grid
CloseDrawing()

```



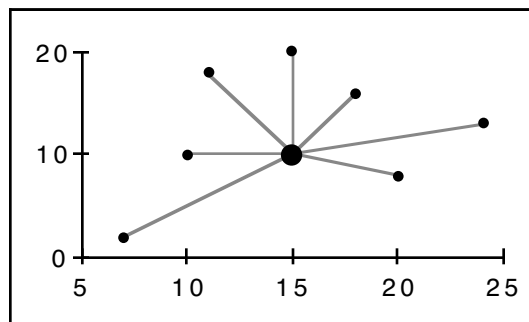


**DropLineReferencePoint**(*seriesIndex*; *xCenter*; *yCenter*;  
*symbolType*; *symbolSize*; *lineWidth*;  
*symbolColor*; *symbolPattern*)

The `DropLineReferencePoint()` function serves to define a reference point which the drop lines radiate from. In this case, as the argument *dropLineAxisIndex* in the `DropLineStyle()` function, all indices greater than 1 are ignored, i.e. only *dropLineAxisIndex*=*x* or *dropLineAxisIndex*=*y* are accepted. The arguments *xCenter* and *yCenter* define the position of the reference point. The appearance of the reference point is controlled by using the arguments *symbolType*, *symbolSize*, *lineWidth*, *symbolColor* and *symbolPattern*. A total of 18 symbols are presently available. An overview of the predefined symbols can be found in *xmReference*. The default reference point is a black, circular point (*symbolType*=*bullet*) with a 9-pixel diameter.

Examples:

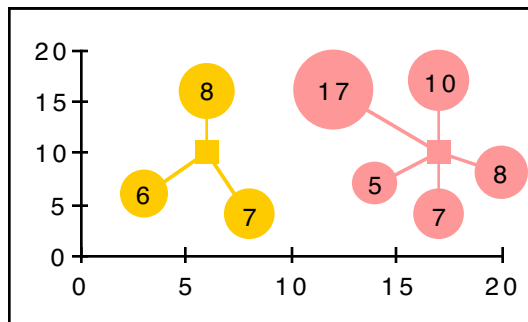
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 7 11 18 20 24 10 15; // x-values
            2 18 16 8 13 10 20) // y-values
  ScatterChart2D()
  SymbolStyle(1;bullet;3;;black)
  DropLineReferencePoint(1;15;10)
  GridLocation(xy;none) // hide grid
CloseDrawing()
```



```

OpenDrawing(200;120)
AddFrame(0;0;200;120)
ChartData(17 14 20 12 17; // 1st series (x-values)
          4 7 8 16 17; // 1st series (y-values)
          7 5 8 17 10; // 1st series (diameters)
          8 3 6; // 2nd series (x-values)
          4 6 16; // 2nd series (y-values)
          7 6 8) // 2nd series (diameters)
BubbleChart2D(label)
BorderStyle(all;none) // hide borders
FillStyle(1;lightRed)
FillStyle(2;darkYellow)
DropLineStyle(1;;1;lightRed)
DropLineStyle(2;;1;darkYellow)
DropLineReferencePoint(1;17;10;square;8;;lightRed)
DropLineReferencePoint(2;6;10;square;8;;darkYellow)
GridLocation(xy;none) // hide grid
CloseDrawing()

```



**DropLineReferenceLine(seriesIndex;xStart;yStart;  
xEnd;yEnd;width;color;pattern)**

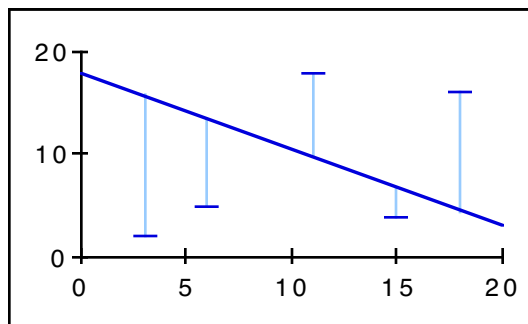
The DropLineReferenceLine() function serves to define a reference line. The arguments *xStart*, *yStart* and *xEnd*, *yEnd* define the position of the line. The reference line is designed by using the arguments *width*, *color* and *pattern*.

## Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 11 18 15 6; // x-values
            2 18 16 4 5) // y-values
  ScatterChart2D()
  SymbolStyle(1;hBar;8;;blue)
  DropLineStyle(1;x;1;lightBlue)
  DropLineReferenceLine(1;0;18;20;3;1;blue)
  GridLocation(all;none) // hide grid
CloseDrawing()

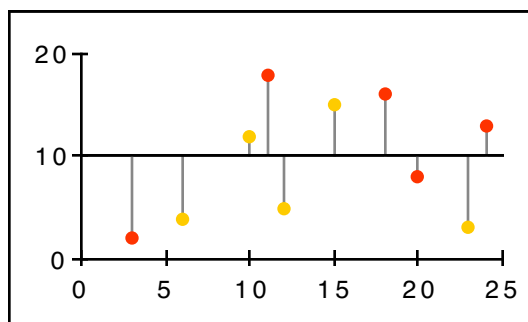
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 3 11 18 20 24; // 1st series (x-values)
            2 18 16 8 13; // 1st series (y-values)
            23 10 15 12 6; // 2nd series (x-values)
            3 12 15 5 4) // 2nd series (y-values)
  ScatterChart2D()
  SymbolStyle(1;bullet;4;;red)
  SymbolStyle(2;bullet;4;;darkYellow)
  DropLineReferenceLine(all;0;10;25;10)
  GridLocation(all;none) // hide grid
CloseDrawing()

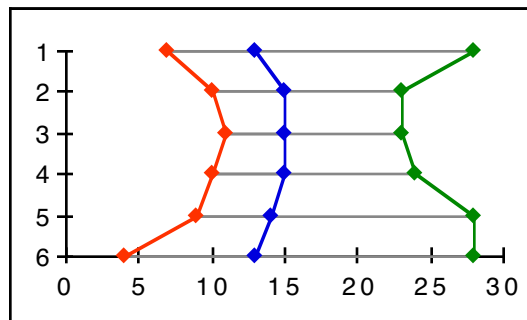
```



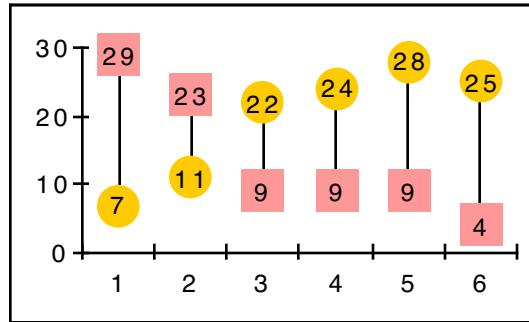
**DropLineReferenceSeries(seriesIndex;refSeriesIndex1;  
refSeriesIndex2...)**

Data points of different series can be connected by drop lines using the DropLineReferenceSeries() function. The "basic series" is defined by the 1st argument *seriesIndex*. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 7 10 11 10 9 4; // 1st series
            13 15 15 15 14 13; // 2nd series
            28 23 23 24 28 28) // 3rd series
  LineChart(horizontal+symbol)
  LineStyle(all;poly;1)
  LineStyle(3;poly;1;green)
  SymbolStyle(all;diamond;4;1)
  SymbolStyle(3;diamond;4;1;green)
  DropLineReferenceSeries(all;1)
  ScalingOptions(y;on) // y-scaling top to bottom
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 7 11 22 24 28 25; // 1st series
            29 23 9 9 9 4) // 2nd series
  ScatterChart(label;on)
  SymbolStyle(1;bullet;15;;darkYellow)
  SymbolStyle(2;square;15;;lightRed)
  LabelOptions(all;centerCenter)
  DropLineStyle(all;x;1;black)
  DropLineReferenceSeries(1;2)
  GridLocation(all;none) // hide grid
CloseDrawing()
```



## Moving Averages

As an option, moving averages can be added to scatter and line charts. A total of three functions are available for this. They make it possible to:

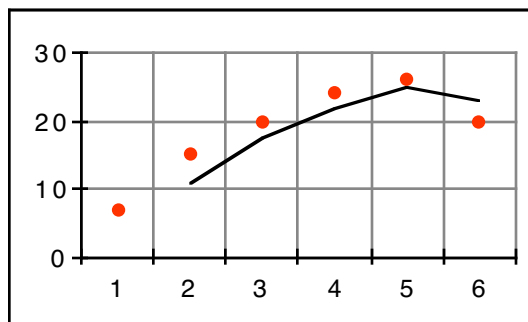
- define different methods of calculation
- design the average lines
- finely tune them using numerous options

For all three functions the series index is defined as the 1st argument and the number of intervals as the 2nd argument. The interval number determines the number of data points used to calculate the average. If, for example, a 50-day average is to be calculated, *numOfIntervals=50* should be set. If no interval number is defined, the average values from two data points will be calculated as the default (*numOfIntervals=2*). If no series index is defined or if *seriesIndex=all* is set, the respective function refers to all available data series.

**MovingAverage(seriesIndex;numOfIntervals;  
calculationMethod;weightList)**

The 1st argument *seriesIndex* in the *MovingAverage()* function defines which data series the moving average is to be calculated for. The 2nd argument *numOfIntervals* defines the number of data points for calculating the average values. Examples:

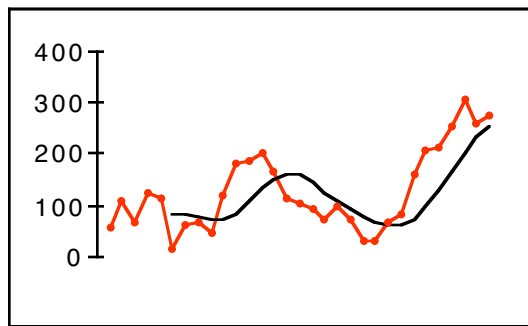
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 15 20 24 26 20)
  ScatterChart(;on)
  SymbolStyle(all;bullet;4;1;red)
  MovingAverage() // moving average of 2 points
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 57 108 70 125 116 13 63 67 45 121 181
            189 201 166 115 105 91 75 99 72 33 29 69
            85 162 210 211 256 304 258 274)
  LineChart(symbol)
  SymbolStyle(1;bullet;2)
  MovingAverage(1;6)      // moving average of 6 points
  AxisOptions(x;none)     // hide x-axis
  GridLocation(all;none)  // hide grid
CloseDrawing()

```

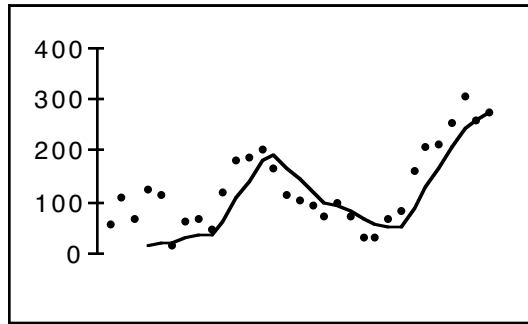


As an option, a list of weighting factors can be entered as the 4th argument. In doing so, the 1st data point is multiplied by the 1st weighting factor, the 2nd data point by the 2nd weighting factor, etc. A weighting factor less than 1 lessens the influence of a data point to form an average; a weighting factor greater than 1 increases the influence. If the number of weighting factors is less than the number of data points, then the neutral value 1 is used in place of the missing factors. The weighting factors are to be entered separated by spaces, tabs or line feeds. Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 57 108 70 125 116 13 63 67 45 121 181
            189 201 166 115 105 91 75 99 72 33 29 69
            85 162 210 211 256 304 258 274)
  ScatterChart()
  SymbolStyle(1;bullet;2;1;black)
  MovingAverage(1;4;;.1 .1 .1 .3 .3 .5 .7 .8 .9 1 1.2)
  AxisOptions(x;none)     // hide x-axis
  GridLocation(all;none)  // hide grid
CloseDrawing()

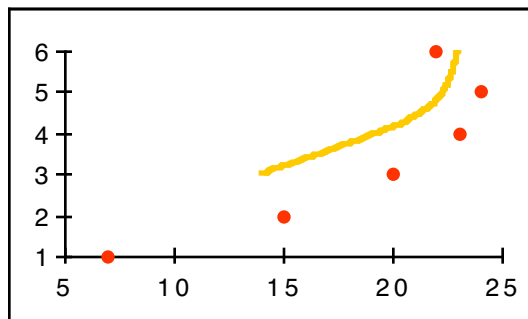
```



**MovingAverageLineStyle(seriesIndex;numOfIntervals;  
calculationMethod;shape;width;color;pattern)**

The `MovingAverageLineStyle()` function can be used to control the appearance of the moving average lines. The 4th argument *shape* controls the shape of the curve — smooth, polygonal or step-like; details can be found in combination with the `LineStyle()` function in the *Styles* section. The line width, color and pattern of the average lines can be varied using the arguments *width*, *color* and *pattern*. Unless otherwise defined, average lines are black, one-pixel wide lines. Example:

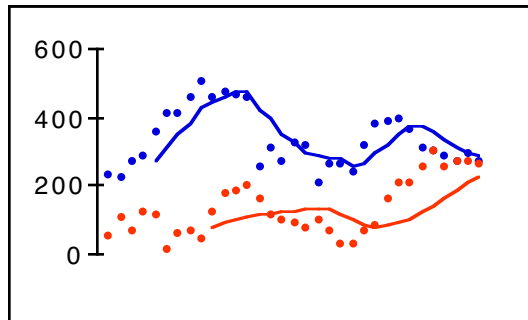
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 15 20 23 24 22)
  ScatterChart(horizontal)
  SymbolStyle(all;bullet;4;1;red)
  MovingAverage(1;3)
  MovingAverageLineStyle(1;3;;smooth;2;darkYellow)
  GridLocation(all;none) // hide grid
CloseDrawing()
```





Currently, three methods are available to calculate moving averages. Details can be found at the `MovingAverageOptions()` function. By calling the `MovingAverage()` function repeatedly, several moving averages can be represented at the same time for both different data series and for a different number of intervals. Examples:

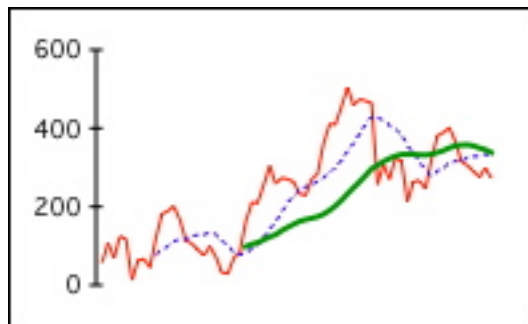
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 57 108 70 125 116 13 63 67 45 121 181
            189 201 166 115 105 91 75 99 72 33 29 69
            85 162 210 211 256 304 258 274 270 263;
            233 229 269 285 362 410 411 456 504 458
            474 470 463 257 308 270 325 316 213 263
            267 245 321 381 389 401 366 315 305 291
            275 299 272)
  ScatterChart()
  SymbolStyle(all;bullet;2)
  MovingAverage(1;10;average)
  MovingAverage(2;5;average)
  MovingAverageLineStyle(1;10;average;poly;1;red)
  MovingAverageLineStyle(2;5;average;poly;1;blue)
  AxisOptions(x;none) // hide x-axis
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 57 108 70 125 116 13 63 67 45 121 181
            189 201 166 115 105 91 75 99 72 33 29 69
            85 162 210 211 256 304 258 274 270 263
            233 229 269 285 362 410 411 456 504 458
            474 470 463 257 308 270 325 316 213 263
            267 245 321 381 389 401 366 315 305 291
            275 299 272)
  LineChart()
  MovingAverage(1;10;average)
  MovingAverage(1;25;average)
  MovingAverageLineStyle(1;10;average;poly;1 2 2;blue)
  MovingAverageLineStyle(1;25;average;poly;2;green)
  AxisOptions(x;none) // hide x-axis
  GridLocation(all;none) // hide grid
CloseDrawing()

```



**MovingAverageOptions(seriesIndex;numOfIntervals;  
calculationMethod;alignment;doExtrapolate;  
hShift;vShift;isRelativeHShift;isRelativeVShift)**

By using the `MovingAverageOptions()` function, calculating and representing moving averages can be varied in many ways. For example, using the 3rd argument *calculationMethod*, three different methods for calculating the average values can be chosen.

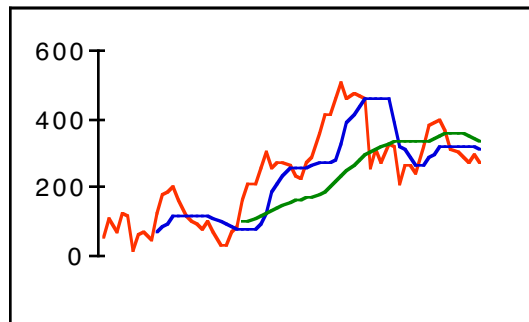
As the default, the average line is calculated using the arithmetic mean values (*calculationMethod=average*). Instead of the mean value, it is also possible to use the median (*calculationMethod=median*).

Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 57 108 70 125 116 13 63 67 45 121 181
            189 201 166 115 105 91 75 99 72 33 29 69
            85 162 210 211 256 304 258 274 270 263
            233 229 269 285 362 410 411 456 504 458
            474 470 463 257 308 270 325 316 213 263
            267 245 321 381 389 401 366 315 305 291
            275 299 272)
  LineChart()
  MovingAverage(1;10;median)
  MovingAverage(1;25;average)
  MovingAverageLineStyle(1;10;median;poly;1;blue)
  MovingAverageLineStyle(1;25;average;poly;1;green)
  MovingAverageOptions(1;10;median)
  MovingAverageOptions(1;25;average)
  AxisOptions(x;none) // hide x-axis
  GridLocation(all;none) // hide grid
CloseDrawing()

```



The 3rd possibility has to do with calculating an exponentially weighted average (*calculationMethod=exponential*) using the formula:

```

w[i+1] = w[i] + a*(y[i+1]-w[i])
with: a.....Smoothing constant between 0 and 1
      y[]...original value
      w[]...exponentially weighted value

```

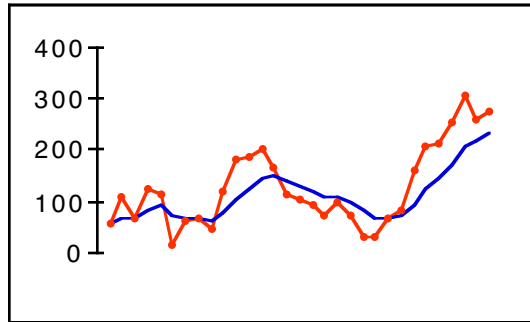
The smoothing constant is entered as the 4th argument in the `MovingAverage()` function instead of the weighting factors. In addition, a start value can also be added to the smoothing constant as an option. In the event of exponential smoothing, the arguments *alignment* and *doExtrapolate* are ignored.

Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(57 108 70 125 116 13 63 67 45 121 181
            189 201 166 115 105 91 75 99 72 33 29
            69 85 162 210 211 256 304 258 274)
  LineChart(symbol)
  SymbolStyle(all;bullet;2)
  // 0.25 = smoothing constant
  MovingAverage(1;;exponential;0.25)
  MovingAverageLineStyle(1;2;exponential;poly;1;blue)
  MovingAverageOptions(1;;exponential)
  AxisOptions(x;none) // hide x-axis
  GridLocation(all;none) // hide grid
CloseDrawing()

```

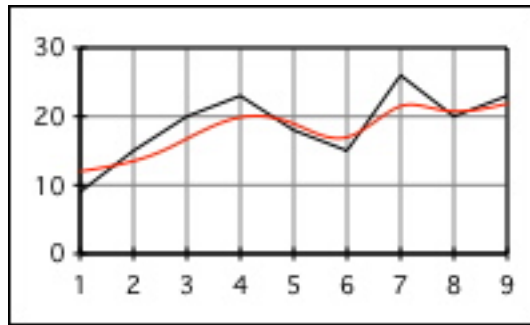


```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(9 15 20 23 18 15 26 20 23)
  LineChart()
  LineStyle(1;poly;1;black)

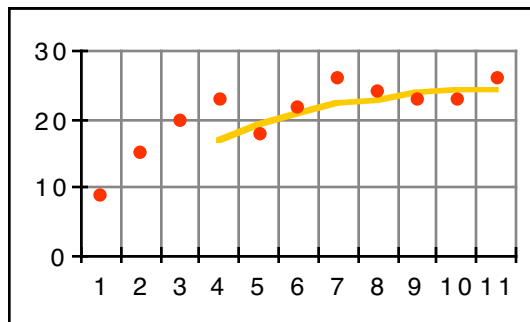
  // 0.5...smoothing constant
  // 15 ...start value
  MovingAverage(1;;exponential;0.5 15)
  MovingAverageLineStyle(1;;exponential;smooth;1;red)
  MovingAverageOptions(1;;exponential)
  GridFrame()
CloseDrawing()

```



By using the 4th argument *alignment*, the position of the average line can be controlled. Average lines — except for exponential smoothing — are always "somewhat shorter" than the original curve since the number of points for drawing the average lines is always *numOfIntervals-1* less than the number of points of the original curve. In other words, for example, a 50-day average line has 49 fewer points than the original curve. Four constants are available for aligning the average lines. As the default, the average line is aligned with the last data point (*alignment=backward*). Example:

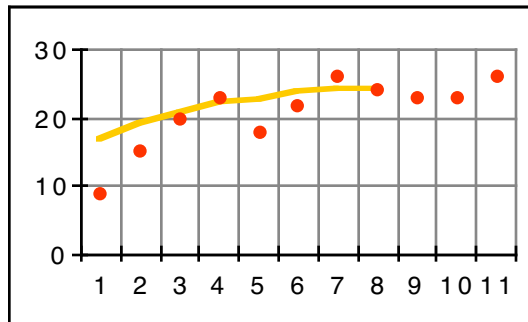
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(9 15 20 23 18 22 26 24 23 23 26)
  ScatterChart(;on)
  SymbolStyle(all;bulet;4;1;red)
  MovingAverage(1;4;average)
  MovingAverageLineStyle(1;4;average;poly;2;darkYellow)
  MovingAverageOptions(1;4;average;backward)
CloseDrawing()
```



The average line is aligned with the first data point if *alignment=forward* is set.

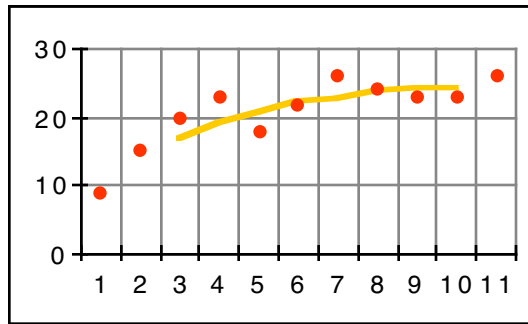
Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(9 15 20 23 18 22 26 24 23 23 26)
  ScatterChart(;on)
  SymbolStyle(all;bullet;4;1;red)
  MovingAverage(1;4;average)
  MovingAverageLineStyle(1;4;average;poly;2;darkYellow)
  MovingAverageOptions(1;4;average;forward)
CloseDrawing()
```



If *alignment=centeredBackward* or *alignment=centeredForward* is set, the average line is centered. That means, for example, for an interval number of 50 and when *alignment=centeredBackward*, the average line is indented 25 points and 24 points when *alignment=centeredForward*. If the number of missing points is even, both constants *centeredForward* and *centeredBackward* produce the same result. Examples:

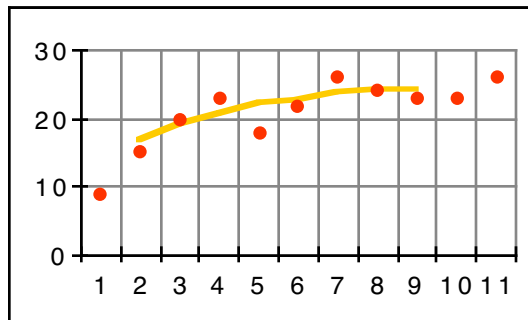
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(9 15 20 23 18 22 26 24 23 23 26)
  ScatterChart(;on)
  SymbolStyle(all;bullet;4;1;red)
  MovingAverage(1;4;average)
  MovingAverageLineStyle(1;4;average;poly;2;darkYellow)
  MovingAverageOptions(1;4;average;centeredBackward)
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(9 15 20 23 18 22 26 24 23 23 26)
  ScatterChart(;on)
  SymbolStyle(all;bullet;4;1;red)
  MovingAverage(1;4;average)
  MovingAverageLineStyle(1;4;average;poly;2;darkYellow)
  MovingAverageOptions(1;4;average;centeredForward)
CloseDrawing()

```

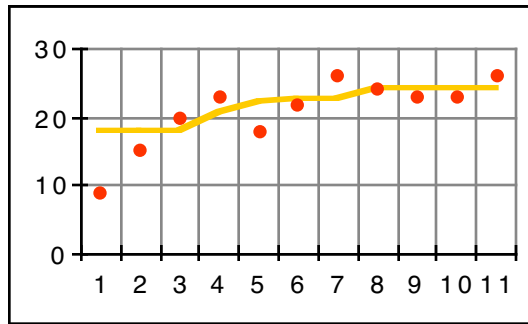


By activating the argument *doExtrapolate=on*, it is possible to extend the average line over the entire data area. Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(9 15 20 23 18 22 26 24 23 23 26)
  ScatterChart(;on)
  SymbolStyle(all;bullet;4;1;red)
  MovingAverage(1;6;average)
  MovingAverageLineStyle(1;6;average;poly;2;darkYellow)
  MovingAverageOptions(1;6;average;centeredForward;on)
CloseDrawing()

```

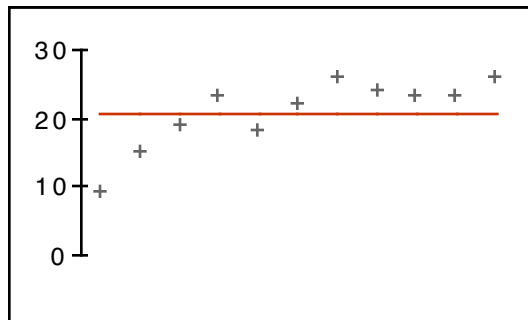


Using *doExtrapolate=on* proves to be advantageous when representing the total average, i.e. the average of the complete data series. Generally, the total average is calculated for a data series consisting of  $n$  values by setting the number of intervals equals  $n$ . Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(9 15 19 23 18 22 26 24 23 23 26)
  ScatterChart(;on)
  SymbolStyle(all;plus;5;1;darkGray)
  MovingAverage(1;11;average)
  MovingAverageLineStyle(1;11;average;poly;1;darkRed)
  MovingAverageOptions(1;11;average;;on)
  AxisOptions(x;none) // hide x-axis
  GridLocation(all;none) // hide grid
CloseDrawing()

```



As an option, moving averages can be shifted by entering a horizontal offset value (*hOffset*) and/or a vertical offset value (*vOffset*). Offset values can either be entered as absolute values (*isRelativeHShift=off* and *isRelativeVShift=off*) (default) or in percent of the average values (*isRelativeHShift=on* and *isRelativeVShift=on*).

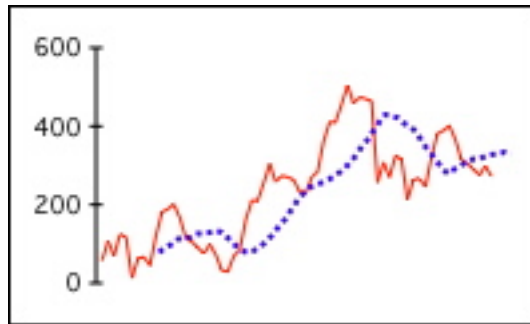


## Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 57 108 70 125 116 13 63 67 45 121 181
            189 201 166 115 105 91 75 99 72 33 29 69
            85 162 210 211 256 304 258 274 270 263
            233 229 269 285 362 410 411 456 504 458
            474 470 463 257 308 270 325 316 213 263
            267 245 321 381 389 401 366 315 305 291
            275 299 272)
  LineChart()
  MovingAverage(1;10;average)
  MovingAverageLineStyle(1;10;average;poly;2 2 2;blue)
  // shift 5% to the right
  MovingAverageOptions(1;10;average;;;5;on)
  AxisOptions(x;none) // hide x-axis
  GridLocation(all;none) // hide grid
CloseDrawing()

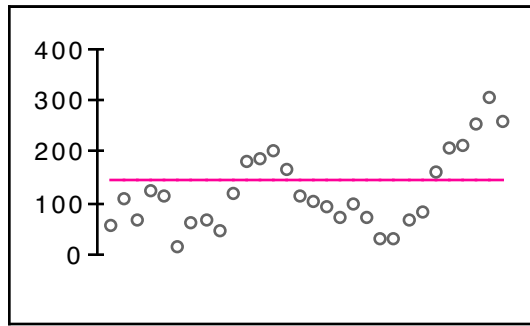
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(57 108 70 125 116 13 63 67
            45 121 181 189 201 166 115
            105 91 75 99 72 33 29 69 85
            162 210 211 256 304 258)
  ScatterChart()
  SymbolStyle(all;circle;4;1;darkGray)
  MovingAverage(1;30;average)
  MovingAverageLineStyle(1;30;average;poly;1;purple)
  // shift 20 units to the top
  MovingAverageOptions(1;30;average;;on;;20)
  AxisOptions(x;none) // hide x-axis
  GridLocation(all;none) // hide grid
CloseDrawing()

```



## Curve Fitting

Three functions are available for fitting curves on scatter and line charts. They make it possible to:

- define different curve functions
- design the curves
- finely tune them using numerous options

For all three functions the series index is defined as the 1st argument and the curve function type as the 2nd argument. If no series index is defined or if *seriesIndex=all* is set, the respective function refers to all available data series. The following curve functions can be chosen:

- Polynomial functions up to the 10th degree. (*type=1...10*)  

$$f(x) = c0 + c1*x + c2*x^2 + c3*x^3 + \dots + cN*x^N$$

The constant *linear* is available in the frequently necessary case of a linear polynomial  $f(x) = c0 + c1*x$  (straight line).
- Power function (*type=pow*):  

$$f(x) = c0 * x^{c1}$$
- Exponential function (*type=exp*):  

$$f(x) = c0 * \exp(c1*x)$$
- Logarithmic function (*type=log*):  

$$f(x) = c0 + c1*\ln(x)$$

The curve coefficients *c0...c10* are calculated by using the method of least squares. By using the external function `xmCH_GetCFValue()`, the correlation coefficient  $r^2$  and the curve coefficients *c0* to max. *c10* can be imported into a FileMaker Pro file. The `xmCH_GetCFValue()` function contains five arguments:

```
xmCH_GetCFValue(chart index;
                 series index;
                 curve function index;
                 value index;
                 format specifier)
```

- Chart index: (required)  
 Since several charts can be drawn at the same time within a drawing, it is necessary to specify the chart by using an index. The index corresponds to the order of chart definitions within `OpenDrawing()` and `CloseDrawing()`. If the drawing only consists of one chart, the chart index is 1.

- **Series index: (required)**  
The series index is used to determine which series the inquiry refers to. This is necessary as curve fittings for several series can be carried out simultaneously.
- **Curve function index: (required)**  
The curve function index is used to define which curve function should be accessed. This is necessary as several curve fittings can be calculated simultaneously for one data series. The curve function index is identical to the argument *type* and has a value between -4 for a logarithmic curve fitting and 10 for a polynomial to the 10th degree.
- **Value index: (required)**  
The value index makes it possible to access individual coefficients. Value index 1 returns the correlation coefficient  $r^2$ , value index 2 the curve coefficient *c0*, value index 3 the curve coefficient *c1*, etc.
- **Format specifier: (optional)**  
As an option, the coefficient returned by the function *xmCH\_GetCFValue()* can be formatted accordingly by using a format specifier. If no format specifier is defined, the default format specifier "|u|" is used. All format specifiers including numerous examples can be found in *xmReference*.

The way in which the external function *xmCH\_GetCFValue()* works can be studied by way of the FileMaker Pro database *xmCFDemo.fp7*. The files are unlocked and are available as a free download.

### **CurveFitting(seriesIndex;type)**

The 1st argument *seriesIndex* in the *CurveFitting()* function defines which data series the curve fitting is to be calculated for. The 2nd argument *type* defines the curve function. The following constants are available for this:

<i>constant</i>	<i>value</i>	<i>function</i>
log	-4	$f(x) = c0 + c1 \cdot \ln(x)$
exp	-3	$f(x) = c0 * \exp(c1 \cdot x)$
pow	-2	$f(x) = c0 * x^{c1}$
linear	1	$f(x) = c0 + c1 \cdot x$ (default)

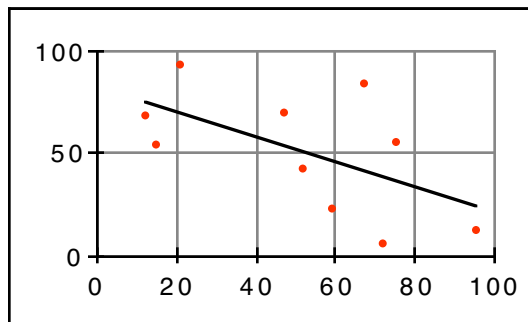
If the 2nd argument *type* is not defined, a 1st degree polynomial (straight line) is calculated.

Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(15 59 72 12 47 21 52 75 67 95; // x-values
            54 23 7 69 70 94 43 56 85 13) // y-values
  ScatterChart2D()
  SymbolStyle(all;bulet;2)
  CurveFitting()
CloseDrawing()

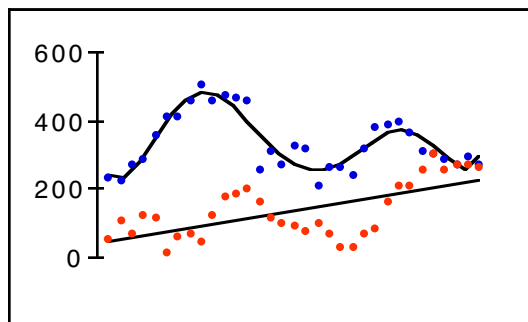
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 57 108 70 125 116 13 63 67 45 121 181
            189 201 166 115 105 91 75 99 72 33 29 69
            85 162 210 211 256 304 258 274 270 263;
            233 229 269 285 362 410 411 456 504 458
            474 470 463 257 308 270 325 316 213 263
            267 245 321 381 389 401 366 315 305 291
            275 299 272)
  ScatterChart()
  SymbolStyle(all;bulet;2)
  CurveFitting(1;1)
  CurveFitting(2;7)
  AxisOptions(x;none) // hide x-axis
  GridLocation(all;none) // hide grid
CloseDrawing()

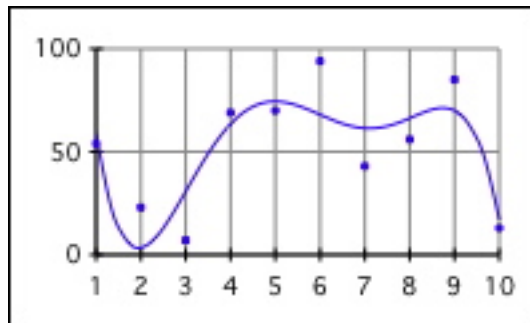
```



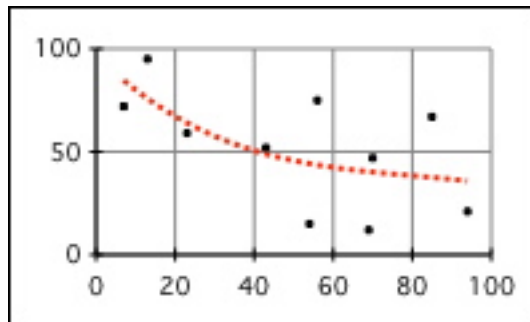
**CurveFittingLineStyle(seriesIndex;type;width;color;pattern)**

The CurveFittingLineStyle() function can be used to control the appearance of the curves. The line width, color and pattern can be varied using the arguments *width*, *color* and *pattern*. Unless otherwise defined, the curves are black and one-pixel wide. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(54 23 7 69 70 94 43 56 85 13)
  ScatterChart()
  SymbolStyle(all;square;2;1;blue)
  CurveFitting(1;5)
  CurveFittingLineStyle(1;5;1;blue)
CloseDrawing()
```



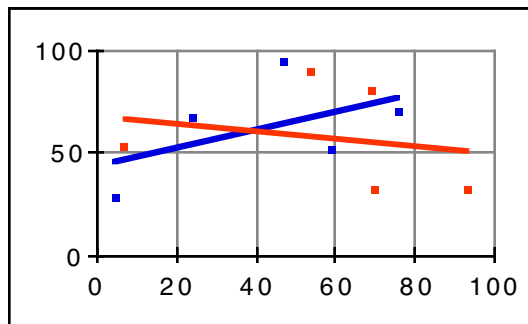
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(54 23 7 69 70 94 43 56 85 13; // x-values
            15 59 72 12 47 21 52 75 67 95) // y-values
  ScatterChart2D()
  SymbolStyle(all;square;2;1;black)
  CurveFitting(1;3)
  CurveFittingLineStyle(1;3;2 2 2;red)
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(54 93 7 69 70; // 1st series (x-values)
            90 33 53 80 33; // 1st series (y-values)
            5 59 76 24 47; // 2nd series (x-values)
            29 52 70 67 95) // 2nd series (y-values)
  ScatterChart2D()
  SymbolStyle(all;square;2)
  CurveFitting(all;1)
  CurveFittingLineStyle(1;1;2;red)
  CurveFittingLineStyle(2;1;2;blue)
CloseDrawing()

```



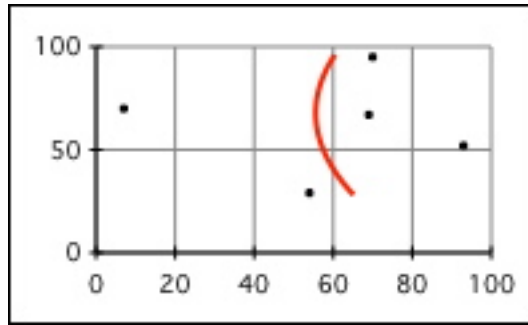
**CurveFittingOptions(seriesIndex;type;doSwitchAxes;  
doExtrapolate;doForceThruZero)**

The CurveFittingOptions() function makes it possible to vary the way in which the curves are calculated and drawn. By activating the argument *doSwitchAxes=on*, the coordinates of the points used to calculate the curves are switched. Example:

```

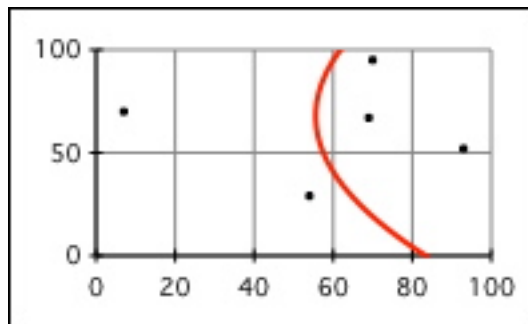
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(54 93 7 69 70; // x-values
            29 52 70 67 95) // y-values
  ScatterChart2D()
  SymbolStyle(all;square;2;1;black)
  CurveFitting(all;2)
  CurveFittingLineStyle(1;2;2;red)
  CurveFittingOptions(all;2;on)
CloseDrawing()

```



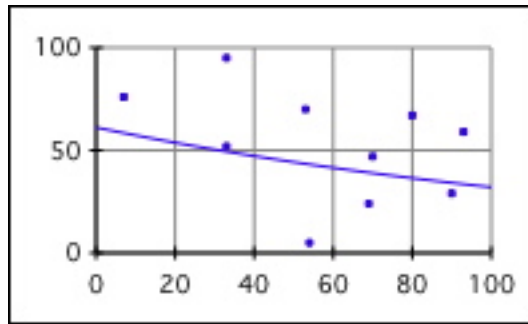
By activating the argument *doExtrapolate=on*, curves can be extended to the border of the graph. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(54 93 7 69 70; // x-values
            29 52 70 67 95) // y-values
  ScatterChart2D()
  SymbolStyle(all;square;2;1;black)
  CurveFitting(all;2)
  CurveFittingLineStyle(1;2;2;red)
  CurveFittingOptions(all;2;on;on)
CloseDrawing()
```



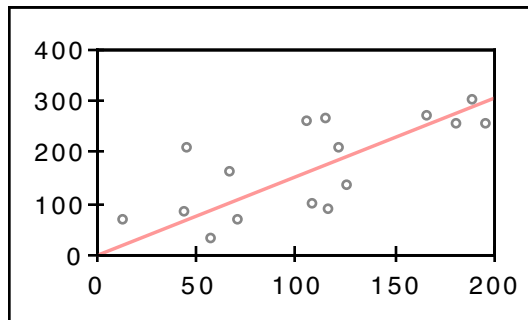
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(54 93 7 69 70 90 33 53 80 33;
            5 59 76 24 47 29 52 70 67 95)
  ScatterChart2D()
  SymbolStyle(all;square;2;1;blue)
  CurveFitting(1;exp)
  CurveFittingLineStyle(1;exp;1;blue)
  CurveFittingOptions(1;exp;;on)
CloseDrawing()
```





Polynomial functions run through the coordinate origin if the argument *doForceThruZero=on* is set. For all other curve functions the argument *doForceThruZero* is ignored. Examples:

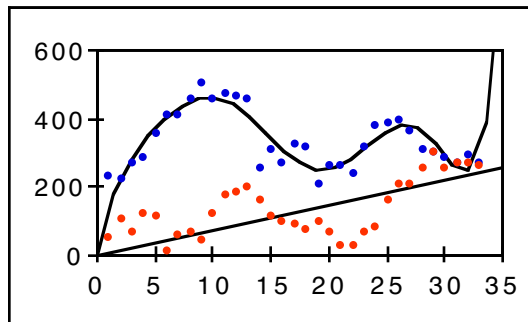
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(57 108 70 125 116 13 43 67 45 121
            181 189 195 166 115 105 91;
            35 99 72 139 90 69 85 162 210 211
            256 304 258 274 270 263)
  ScatterChart2D()
  SymbolStyle(all;circle;3;1;gray)
  CurveFitting(1;linear)
  CurveFittingLineStyle(1;linear;1;lightRed)
  CurveFittingOptions(1;linear;;on;on)
  MajorGridLineWidths(all;all;0) // hide grid
  GridLocation(xy;front)
  GridFrame()
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 57 108 70 125 116 13 63 67 45 121 181
            189 201 166 115 105 91 75 99 72 33 29 69
            85 162 210 211 256 304 258 274 270 263;
            233 229 269 285 362 410 411 456 504 458
            474 470 463 257 308 270 325 316 213 263
            267 245 321 381 389 401 366 315 305 291
            275 299 272)
  ScatterChart()
  SymbolStyle(all;bullet;2)
  CurveFitting(1;linear)
  CurveFitting(2;7)
  CurveFittingOptions(1;1;;on;on)
  CurveFittingOptions(2;7;;on;on)
  MajorGridLineWidths(all;all;0) // hide grid
  GridFrame()
CloseDrawing()

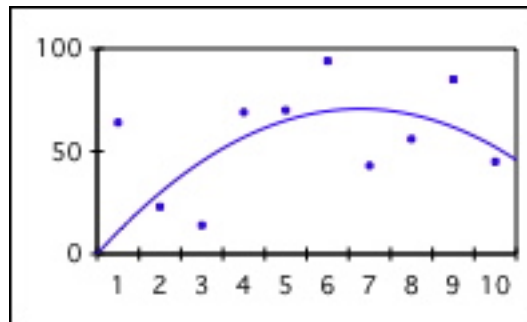
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(64 23 14 69 70 94 43 56 85 45)
  ScatterChart(;on)
  SymbolStyle(all;square;2;1;blue)
  CurveFitting(1;2)
  CurveFittingLineStyle(1;2;1;blue)
  CurveFittingOptions(1;2;off;on;on)
  MajorGridLineWidths(all;all;0) // hide grid
  GridFrame()
CloseDrawing()

```



## Error Bars

Error bars can be added to scatter, line and bar charts. Four functions are available for calculating and drawing error bars.

**ErrorBars(seriesIndex;axisIndex;errorDirection;type;  
addlData1;addlData2;addlData3;addlData4)**

By using the `ErrorBars()` function, all important parameters for calculating error bars can be controlled. The 1st argument *seriesIndex* defines which data series the function refers to. By using the 2nd argument *axisIndex*, the direction of the error bars can be defined for two-dimensional charts. If no axis index is defined or if *axisIndex=all* is set, error bars are calculated in both the x and y-direction. By using the 3rd argument *errorDirection*, it is possible to define whether only positive errors (*errorDirection=plus*) or only negative errors (*errorDirection=minus*) or both errors (*errorDirection=both*) should be represented. As the default, both errors are represented (*errorDirection=both*). The method of error calculation is defined by using the argument *type*. The following error calculations are possible:

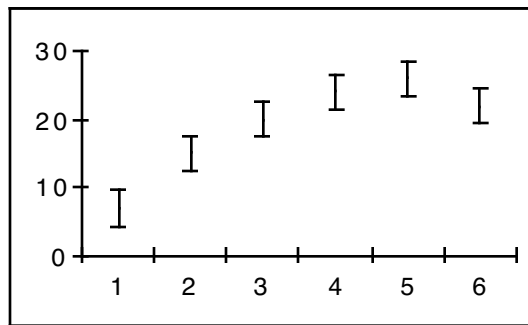
- Standard error: (*type=stdError*)
- Standard deviation: (*type=stdDev*)
- Percent error: (*type=percent*)
- Constant error: (*type=constant*)
- User-defined error lists: (*type=valueList*)

If *type* is not defined, the standard error is calculated by default. User-defined error lists (*type=valueList*) are entered by using the `ErrorBarData()` function. The meaning of the arguments *addlData1* to *addlData4* depends on the type of error calculation:

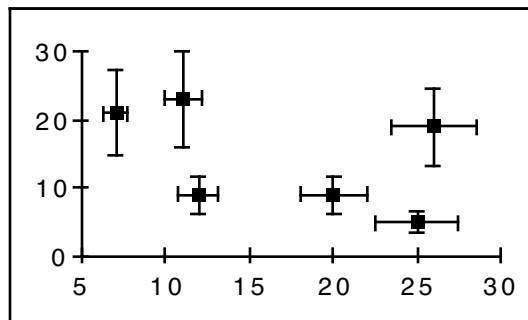
- Standard error:  
*addlData1...addlData4* have no meaning and are ignored.
- Standard deviation:  
*addlData1...addlData4* are standard deviations, default: 1
- Percent error:  
*addlData1...addlData4* are percent values, default: 5 [%]
- Constant error:  
*addlData1...addlData4* are constant values, default: 1
- User-defined error lists:  
*addlData1...addlData4* have no meaning and are ignored.

Argument *addData1* refers to the positive error in the x-direction, argument *addData2* to the negative error in the x-direction; arguments *addData3* and *addData4* refer to the positive and negative errors in the y-direction. Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 15 20 24 26 22)
  ScatterChart(;on)
  SymbolStyle(all;none)
  ErrorBars() // default: standard error
  GridLocation(all;none) // hide grid
CloseDrawing()
```



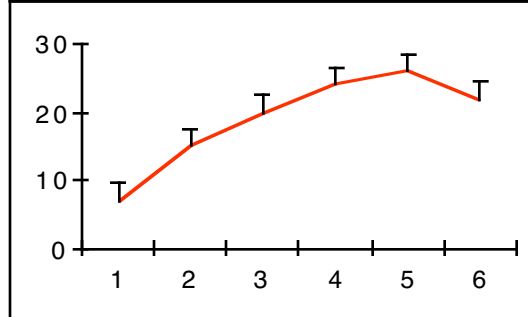
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 7 11 12 20 26 25; // x-values
            21 23 9 9 19 5) // y-values
  ScatterChart2D()
  SymbolStyle(all;square;4;1;black)
  ErrorBars(1;all;both;percent;10;10;30;30)
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 15 20 24 26 22)
  LineChart(;on)
  ErrorBars(1;;plus)      // default: standard error
  GridLocation(all;none) // hide grid
CloseDrawing()

```



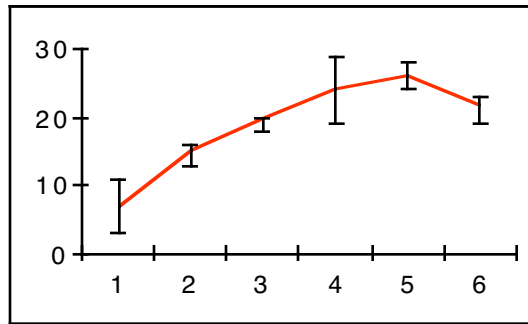
**ErrorBarData(seriesIndex;valueList1;valueList2;  
valueList3;valueList4)**

By using the ErrorBarData() function, user-defined error values can be entered. Correspondingly, the 4th argument *type=valueList* should be set in the ErrorBars() function; otherwise, the ErrorBarData() function is ignored. Argument *valueList1* contains the positive error values in the x-direction; argument *valueList2* the negative error values in the x-direction. Arguments *valueList3* and *valueList4* represent the positive and negative error values in the y-direction. The list values are to be entered separated by spaces, tabs or line feeds. If the number of error values entered is smaller than the number of data points, then zeroes are used in place of the missing values. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 15 20 24 26 22)
  LineChart(;on)
  ErrorBars(1;y:both;valueList)
  ErrorBarData(1;;;4 1 0 5 2 1;4 2 2 5 2 3)
  GridLocation(all;none) // hide grid
CloseDrawing()

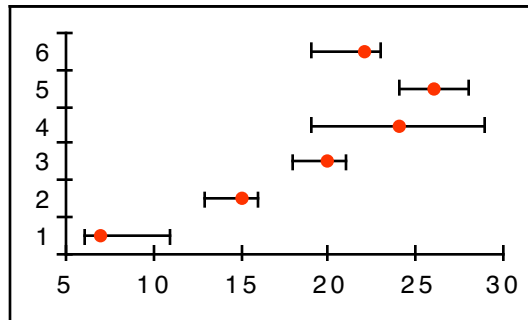
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 15 20 24 26 22)
  ScatterChart(horizontal;on)
  SymbolStyle(all;bullet;4;1;red)
  ErrorBars(1;;;valueList)
  ErrorBarData(1;4 1 1 5 2 1; 1 2 2 5 2 3)
  GridLocation(all;none) // hide grid
CloseDrawing()

```

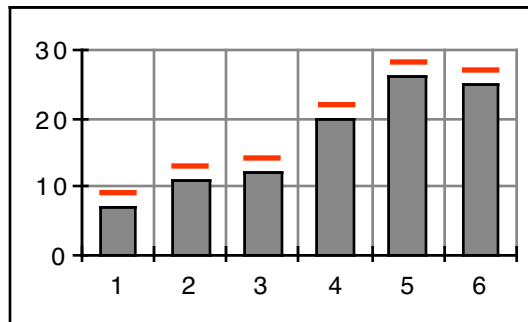


**ErrorBarStyle(seriesIndex;axisIndex;showCapsOnly;  
capLength;barWidth;barColor;barPattern)**

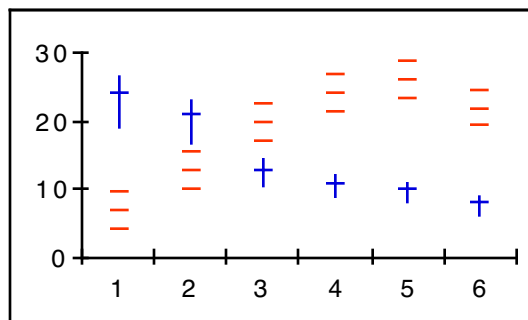
The appearance of the error bars can be controlled by using the `ErrorBarStyle()` function. The 1st argument *seriesIndex* defines which data series the function refers to. The 2nd argument *axisIndex* defines which error bar direction the function refers to. By activating the argument *showCapsOnly=on*, the connecting lines between the error markers can be suppressed. The length of the error markers (in pixels) can be controlled by using the argument *capLength*. The line width, color and pattern can be varied using the arguments *barWidth*, *barColor* and *barPattern*. Unless otherwise defined, error bars are black and one-pixel wide.

Examples:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(7 11 12 20 26 25)
  BarChart()
  FillStyle(all;gray)
  ErrorBars(1;all;plus;constant;;;2)
  ErrorBarStyle(all;;on;13;2;red)
CloseDrawing()
```



```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 7 13 20 24 26 22; // 1st series
            24 21 13 11 10 8) // 2nd series
  ScatterChart(;on)
  SymbolStyle(all;hBar;6)
  ErrorBars(1;y:both;stdError)
  ErrorBars(2;y:both;percent;;;10;20)
  ErrorBarStyle(1;y;on ;6;1;red)
  ErrorBarStyle(2;y;off;0;1;blue)
  GridLocation(all;none) // hide grid
CloseDrawing()
```

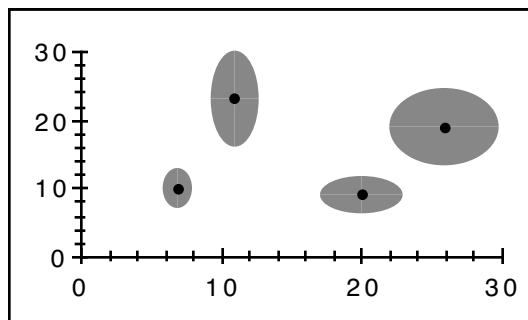




**ErrorBarStyle2D(seriesIndex;shapeType;fillColor;  
fillPattern;borderWidth;borderColor;  
borderPattern)**

By using the `ErrorBarStyle2D()` function, the error values on two-dimensional charts can also be represented by ellipses or rectangles. The argument *shapeType* makes it possible to choose between elliptical error areas (*shapeType=oval*) or rectangular error areas (*shapeType=rect*). If the form is not defined, ellipses are drawn as the default to represent the errors. The area can be varied by using the arguments *fillColor* and *fillPattern*, the border by using the arguments *borderWidth*, *borderColor* and *borderPattern*. As the default, the ellipses are drawn with a one-pixel wide, black border and are not filled. Examples:

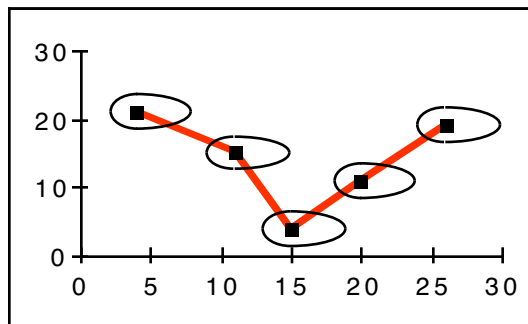
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 7 11 20 26; // x-values
            10 23 9 19) // y-values
  ScatterChart2D()
  Scaling(all;linear;0;30;3;5)
  SymbolStyle(all;bullet;3;1;black)
  ErrorBars(1;all;both;percent;15;15;30;30)
  ErrorBarStyle(1;all;on;0)
  ErrorBarStyle2D(1;oval;gray;black;0)
  GridLocation(all;none) // hide grid
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(4 11 15 20 26; 21 15 4 11 19)
  LineChart2D(symbol)
  LineStyle(1;poly;2)
  SymbolStyle(all;square;4;1;black)
  ErrorBars(1;x:both;constant;4;2)
  ErrorBars(1;y:both;stdError)
  ErrorBarStyle(1;all;on;0)
  ErrorBarStyle2D(1;oval)
  GridLocation(all;none) // hide grid
CloseDrawing()

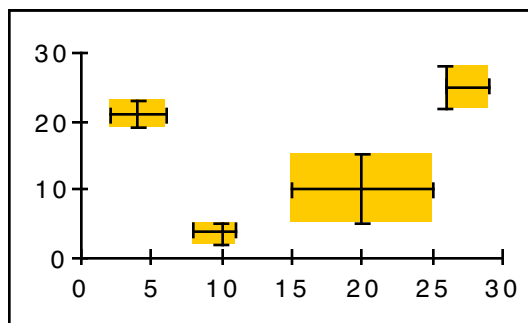
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData( 4 10 26 20; // x-values
            21  4 25 10) // y-values
  ScatterChart2D()
  SymbolStyle(all;none;3;1;black)
  ErrorBars(1;all:both;valueList)
  ErrorBarData(1;2 1 3 5; // positive x-errors
               2 2 0 5; // negative x-errors
               2 1 3 5; // positive y-errors
               2 2 3 5) // negative y-errors
  ErrorBarStyle2D(1;rect;darkYellow;black;0)
  GridLocation(all;none) // hide grid
CloseDrawing()

```



## Graphic Primitives

In addition to the chart functions, a set of over 20 graphic primitives for drawing lines, rectangles, ellipses, polygons, paths, bitmaps, texts, etc. is available which makes it possible to add texts, logos, etc to charts. Even complete drawings can be created using the primitives. The power and variety of the graphic primitives make it possible to use them in a great many ways.

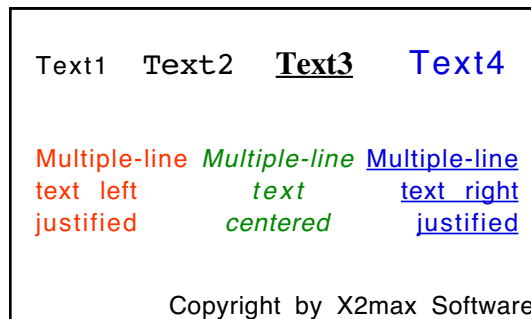
For all primitives the coordinates refer to the upper left-hand corner of the drawing. If functions are entered within a view, the coordinates refer to the upper left-hand corner of the view. Views can also be used to combine several functions into groups. The groups can then simply be shifted by changing the position of the views.

```
AddText(hPosition;vPosition;text;font;size;style;  
color;hAlignment;vAlignment;orientation;  
maxWidth;maxHeight;ellipsisPosition)
```

For left-justified texts (default) the left border of the text is defined by *hPosition*, for centered texts (*hAlignment=center*) *hPosition* defines the center of the text and for right-justified texts (*hAlignment=right*) *hPosition* defines the right border of the text. By using the argument *vPosition*, the vertical position of the first line of text can be defined. If *vAlignment=baseline* is set (default), then *vPosition* refers to the base line of the first text line. When *vAlignment=top*, the upper border of the text is defined by *vPosition*; when *vAlignment=center*, *vPosition* defines the vertical center of the text; when *vAlignment=bottom*, *vPosition* defines the lower border of the text.

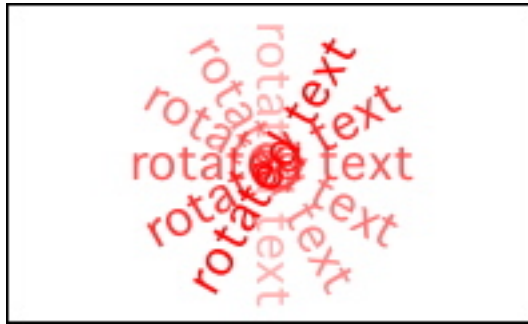
Texts consisting of several lines can be created by inserting a line feed "\n". Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddText( 10;25;"Text1")
  AddText( 50;25;"Text2";"Courier";12)
  AddText(100;25;"Text3";"Times";12:bold+underline)
  AddText(150;25;"Text4";;12;plain;blue)
  AddText(10;60;"Multiple-line\n text left\n justified";
    ;;;red)
  AddText(100;60;"Multiple-line\n text\n centered";
    ;;italic;green;center)
  AddText(190;60;"Multiple-line\n text right\n justified";
    ;;underline;blue;right)
  AddText(60;115;"Copyright by X2max Software")
CloseDrawing()
```



When *orientation*=0 (in degrees), the text is laid out horizontally (default). When *orientation*>0, the text is turned clockwise; when *orientation*<0, it is turned counterclockwise. Example:

```
OpenDrawing(200;120)
AddFrame(0;0;200;120)
AddText(100;60;"rotated text";;18;;255 0 0 255;2;2;-60)
AddText(100;60;"rotated text";;18;;255 0 0 210;2;2;-30)
AddText(100;60;"rotated text";;18;;255 0 0 170;2;2; 0)
AddText(100;60;"rotated text";;18;;255 0 0 140;2;2; 30)
AddText(100;60;"rotated text";;18;;255 0 0 120;2;2; 60)
AddText(100;60;"rotated text";;18;;255 0 0 80;2;2; 90)
CloseDrawing()
```



When setting a maximum width (*maxWidth*>0), a longer multi-line text can be issued as continuous text with automatic word wraparound. The height of the text block can be controlled by the argument *maxHeight*. Texts which do not completely fit into the area predefined by *maxWidth* and *maxHeight* will be shortened according to the setting by the argument *ellipsisPosition*. Five options are available:

*ellipsisPosition*=0: Text is clipped at the end, no "..."

*ellipsisPosition*=1: Text is clipped in the beginning (...text)

*ellipsisPosition*=2: Text is clipped in the middle (text...text)

*ellipsisPosition*=3: Text is clipped at the end (text...) – default

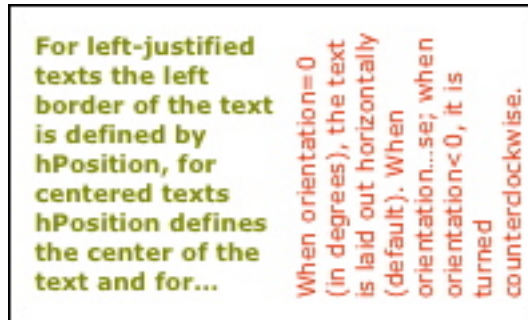
*ellipsisPosition*=4: Text is clipped in the beginning + at the end (...text...)

Example:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddText(10;10;"For left-justified texts the left
border of the text is defined by hPosition, for centered
texts hPosition defines the center of the text and for right-
justified texts hPosition defines the right border of the
text.");
    "Verdana";9;bold;olive;left;top;0;90;100)
  AddText(100;25;"When orientation=0 (in degrees), the
text is laid out horizontally (default). When orientation>0,
the text is turned clockwise; when orientation<0, it is
turned counterclockwise.");
    "Verdana";9;plain;darkRed;left;;-90;100;90;2)
CloseDrawing()

```

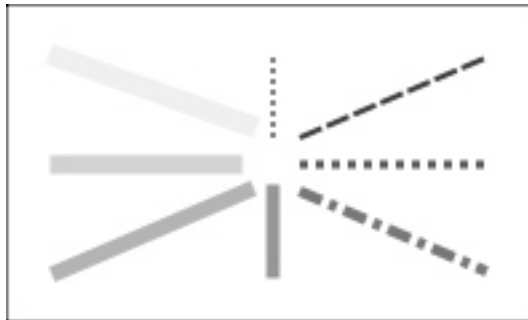


As the default, *maxWidth=-1* and *maxHeight=-1* are set, i.e. no width and height constraints and, thus, no automatic word wraparound support.

**AddLine(hStart;vStart;hEnd;vEnd;lineWidth;lineColor;  
linePattern)**

Lines can be added by using the AddLine() function whose line width, color and pattern can be varied. The coordinate origin is located in the upper left-hand corner of the drawing or view. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddLine(100;50;100; 20;1 2 2; 0 0 0)
  AddLine(110;50;180; 20;2 9 2; 60 60 60)
  AddLine(110;60;180; 60;3 3 3; 90 90 90)
  AddLine(110;70;180;100;4 9 3 3 3;120 120 120)
  AddLine(100;70;100;100;5;150 150 150)
  AddLine( 90;70; 20;100;6;180 180 180)
  AddLine( 85;60; 20; 60;7;210 210 210)
  AddLine( 90;45; 20; 20;8;240 240 240)
CloseDrawing()
```



**AddFrame(left;top;width;height;frameWidth;frameColor;  
framePattern)**

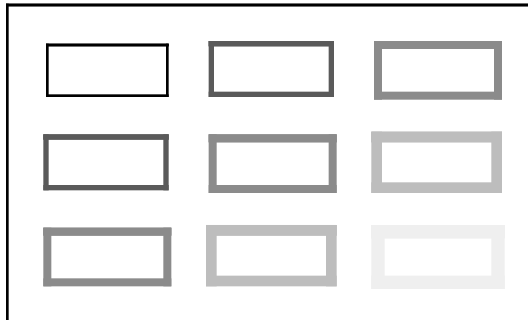
The AddFrame() function makes it possible to create a rectangular frame whose width, color and pattern can be varied. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)

  // left column
  AddFrame(15;15;46;20;1; 0 0 0)
  AddFrame(15;50;46;20;2; 90 90 90)
  AddFrame(15;85;46;20;3;140 140 140)

  // center column
  AddFrame(77;15;46;20;2; 90 90 90)
  AddFrame(77;50;46;20;3;140 140 140)
  AddFrame(77;85;46;20;4;190 190 190)

  // right column
  AddFrame(139;15;46;20;3;140 140 140)
  AddFrame(139;50;46;20;4;190 190 190)
  AddFrame(139;85;46;20;5;240 240 240)
CloseDrawing()
```





**AddRect(left;top;width;height;fillColor;fillPattern)**

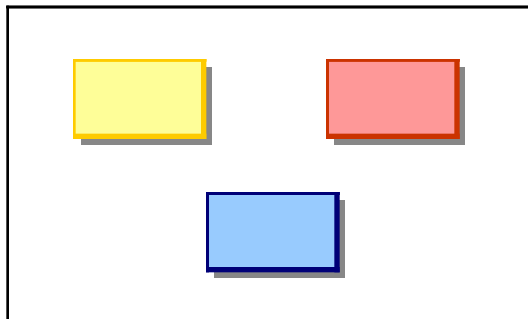
The AddRect() function draws rectangular areas whose color and pattern can be varied. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)

  OpenView(25;20;53;33) // top left
    AddRect(3;3;50;30;gray)
    AddRect(0;0;50;30;lightYellow)
    AddFrame(0;0;50;30;2;darkYellow)
  CloseView()

  OpenView(120;20;53;33) // top right
    AddRect(3;3;50;30;gray)
    AddRect(0;0;50;30;lightRed)
    AddFrame(0;0;50;30;2;darkRed)
  CloseView()

  OpenView(75;70;53;33) // bottom center
    AddRect(3;3;50;30;gray)
    AddRect(0;0;50;30;lightBlue)
    AddFrame(0;0;50;30;2;darkBlue)
  CloseView()
CloseDrawing()
```



```
AddRoundFrame(left;top;width;height;hCurvature;  
vCurvature;frameWidth;frameColor;  
framePattern)
```

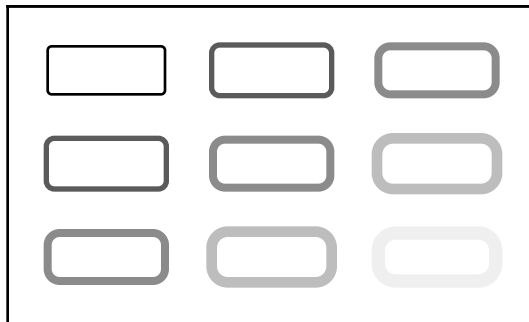
The `AddRoundFrame()` function is used to create frames with rounded corners whose width, color and pattern can be varied. The curvature is controlled by using the arguments *hCurvature* and *vCurvature*. The greater *hCurvature* and *vCurvature*, the greater the rounding of the corner. When *hCurvature*=0 and *vCurvature*=0, the rounding disappears and `AddRoundFrame()` changes to `AddFrame()`. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)

  // left column
  AddRoundFrame(15;15;46;20; 4; 4;1; 0 0 0)
  AddRoundFrame(15;50;46;20; 8; 8;2; 90 90 90)
  AddRoundFrame(15;85;46;20;12;12;3;140 140 140)

  // center column
  AddRoundFrame(77;15;46;20; 8; 8;2; 90 90 90)
  AddRoundFrame(77;50;46;20;12;12;3;140 140 140)
  AddRoundFrame(77;85;46;20;16;16;4;190 190 190)

  // right column
  AddRoundFrame(139;15;46;20;12;12;3;140 140 140)
  AddRoundFrame(139;50;46;20;16;16;4;190 190 190)
  AddRoundFrame(139;85;46;20;20;20;5;240 240 240)
CloseDrawing()
```



**AddRoundRect(left;top;width;height;hCurvature;  
vCurvature;fillColor;fillPattern)**

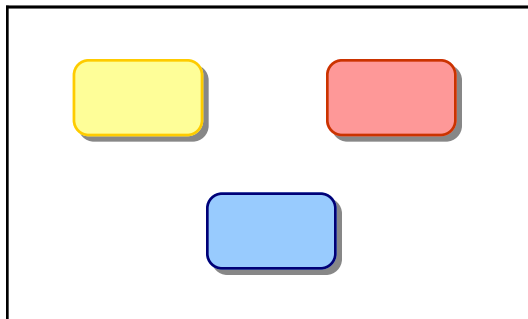
The AddRoundRect() function makes it possible to draw rectangles with rounded corners whose color and pattern can be varied. The curvature is controlled by using the arguments *hCurvature* and *vCurvature*. The greater *hCurvature* and *vCurvature*, the greater the rounding of the corner. When *hCurvature*=0 and *vCurvature*=0, the rounding disappears and AddRoundRect() changes to AddRect(). Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)

  OpenView(25;20;52;32) // top left
    AddRoundRect(2;2;50;30;;;gray)
    AddRoundRect(0;0;50;30;;;lightYellow)
    AddRoundFrame(0;0;50;30;;;1;darkYellow)
  CloseView()

  OpenView(120;20;52;32) // top right
    AddRoundRect(2;2;50;30;;;gray)
    AddRoundRect(0;0;50;30;;;lightRed)
    AddRoundFrame(0;0;50;30;;;1;darkRed)
  CloseView()

  OpenView(75;70;52;32) // bottom center
    AddRoundRect(2;2;50;30;;;gray)
    AddRoundRect(0;0;50;30;;;lightBlue)
    AddRoundFrame(0;0;50;30;;;1;darkBlue)
  CloseView()
CloseDrawing()
```



**AddEllipse(left;top;width;height;lineWidth;lineColor;  
linePattern)**

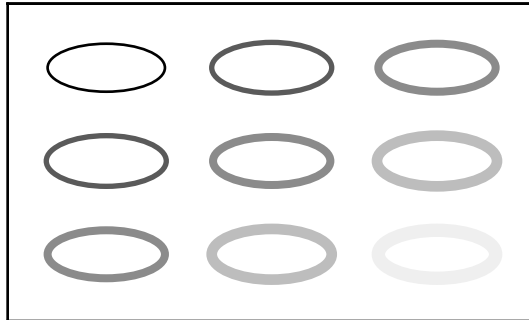
The AddEllipse() function makes it possible to create oval frames whose width, color and pattern can be varied. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)

  // left column
  AddEllipse(15;15;46;20;1; 0  0  0)
  AddEllipse(15;50;46;20;2; 90 90 90)
  AddEllipse(15;85;46;20;3;140 140 140)

  // center column
  AddEllipse(77;15;46;20;2; 90 90 90)
  AddEllipse(77;50;46;20;3;140 140 140)
  AddEllipse(77;85;46;20;4;190 190 190)

  // right column
  AddEllipse(139;15;46;20;3;140 140 140)
  AddEllipse(139;50;46;20;4;190 190 190)
  AddEllipse(139;85;46;20;5;240 240 240)
CloseDrawing()
```



**AddOval(left;top;width;height;fillColor;fillPattern)**

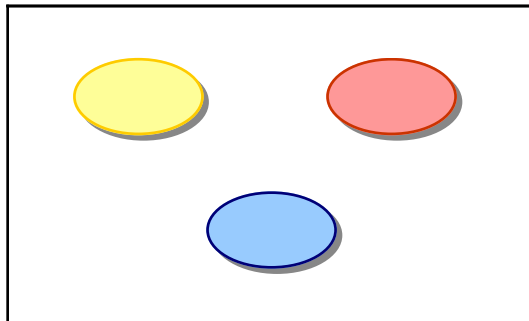
The AddOval() function makes it possible to create oval areas whose color and pattern can be varied. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)

  OpenView(25;20;52;32) // top left
    AddOval(2;2;50;30;gray)
    AddOval(0;0;50;30;lightYellow)
    AddEllipse(0;0;50;30;1;darkYellow)
  CloseView()

  OpenView(120;20;52;32) // top right
    AddOval(2;2;50;30;gray)
    AddOval(0;0;50;30;lightRed)
    AddEllipse(0;0;50;30;1;darkRed)
  CloseView()

  OpenView(75;70;53;33) // bottom center
    AddOval(2;2;50;30;gray)
    AddOval(0;0;50;30;lightBlue)
    AddEllipse(0;0;50;30;1;darkBlue)
  CloseView()
CloseDrawing()
```



**AddArc(left;top;width;height;startAngle;arcAngle;  
lineWidth;lineColor;linePattern)**

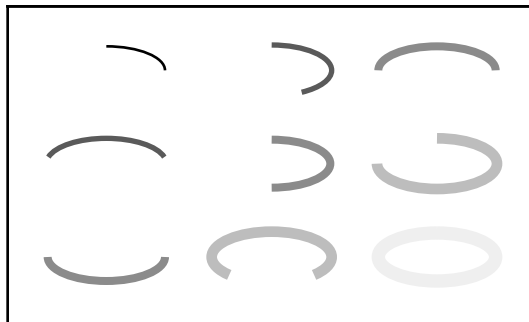
The AddArc() function makes it possible to create oval arcs whose width, color and pattern can be varied. The start and arc angles are to be entered within a range of  $\pm 360$  degrees; *startAngle*=0 degrees is at the top (12 o'clock), *startAngle*=90 degrees is on the right (3 o'clock) and *startAngle*=-90 degrees is on the left (9 o'clock). If the arc angle is positive, the arc is drawn clockwise; if the arc angle is negative, the arc is drawn counterclockwise. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)

  // left column
  AddArc(15;15;46;20; 0; 90;1; 0 0 0)
  AddArc(15;50;46;20;-75;150;2; 90 90 90)
  AddArc(15;85;46;20; 90;180;3;140 140 140)

  // center column
  AddArc(77;15;46;20; 0;150;2; 90 90 90)
  AddArc(77;50;46;20; 0;180;3;140 140 140)
  AddArc(77;85;46;20;-135;270;4;190 190 190)

  // right column
  AddArc(139;15;46;20;-90;180;3;140 140 140)
  AddArc(139;50;46;20; 0;270;4;190 190 190)
  AddArc(139;85;46;20; 0;360;5;240 240 240)
CloseDrawing()
```



**AddSlice(left;top;width;height;startAngle;arcAngle;  
innerRadius;fillColor;fillPattern)**

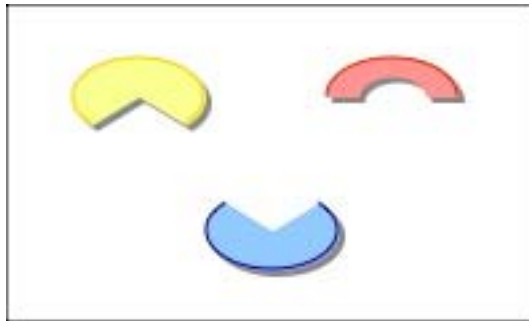
The AddSlice() function is used to draw oval slices whose color and pattern can be varied. The arguments *startAngle* and *arcAngle* are explained in combination with the AddArc() function. The argument *innerRadius* makes it possible to draw ring-shaped sectors. The inner radius is to be entered in percentage of the half axis — between 0 (default) and 100. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)

  OpenView(25;20;52;32) // top left
    AddSlice(2;2;50;30;-135;270;;gray)
    AddSlice(0;0;50;30;-135;270;;lightYellow)
    AddArc(0;0;50;30;-135;270;1;darkYellow)
  CloseView()

  OpenView(120;20;52;32) // top right
    AddSlice(2;2;50;30;-90;180;50;gray)
    AddSlice(0;0;50;30;-90;180;50;lightRed)
    AddArc(0;0;50;30;-90;180;1;darkRed)
  CloseView()

  OpenView(75;70;53;33) // bottom center
    AddSlice(2;2;50;30;45;270;;gray)
    AddSlice(0;0;50;30;45;270;;lightBlue)
    AddArc(0;0;50;30;45;270;1;darkBlue)
  CloseView()
CloseDrawing()
```



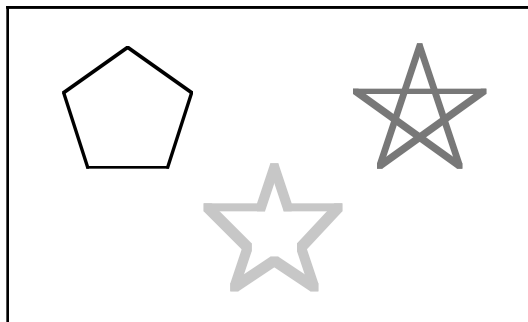
**AddPolyline(scanDirection;listOfCoords;lineWidth;  
lineColor;linePattern)**

The AddPolyline() function makes it possible to draw continuous straight lines whose line width, color and pattern can be varied. The argument *scanDirection* makes it possible to enter coordinates in two ways: either all x-coordinates are entered first and then all y-coordinates (*scanDirection=xyxy*) or the x and y-coordinates of the 1st polygon point are entered, then the x and y-coordinates of the 2nd polygon point, etc. (*scanDirection=xyxy*). All coordinate values are to be entered separated by spaces, tabs or line feeds. The following two functions define the same polygon:

```
(1) AddPolyline(xyxy;45 69 60 30 21 45
                    15 32 60 60 32 15)
(2) AddPolyline(xyxy;45 15
                    69 32
                    60 60
                    30 60
                    21 32
                    45 15)
```

Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddPolyline(xyxy;45 15 69 32 60 60 30 60
              21 32 45 15; 1)
  AddPolyline(xyxy;155 15 140 60 179 32 131 32
              170 60 155 15; 2;120 120 120)
  AddPolyline(xyxy;100 60 95 75 75 75 90 90
              85 105 100 95 115 105 110 90
              124 75 105 75 100 60;
              3;200 200 200)
CloseDrawing()
```

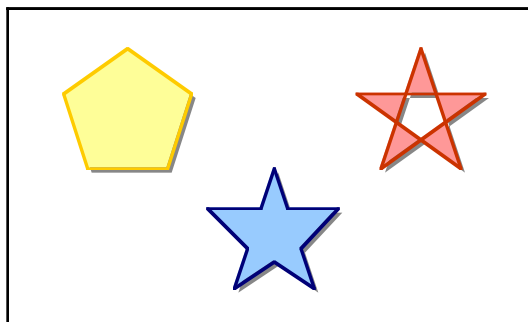




**AddPolygon(scanDirection;listOfCoords;fillColor;  
fillPattern)**

The AddPolygon() function makes it possible to draw polygonal areas whose color and pattern can be varied. The arguments *scanDirection* and *listOfCoords* are explained in combination with the AddPolyline() function. Example:

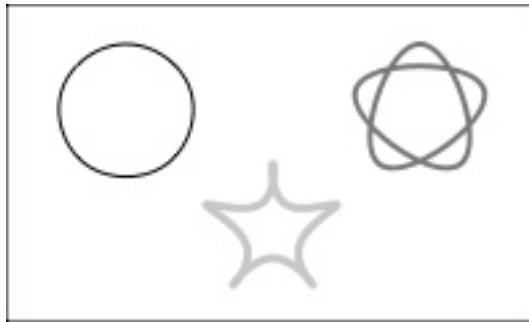
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddPolygon(xyxy;47 17 71 34 62 62 32 62
             23 34 47 17; gray)
  AddPolygon(xyxy;45 15 69 32 60 60 30 60
             21 32 45 15; lightYellow)
  AddPolyline(xyxy;45 15 69 32 60 60 30 60
              21 32 45 15; 1; darkYellow)
  AddPolygon(xyxy;157 17 142 62 181 34 133 34
             172 62 157 17; gray)
  AddPolygon(xyxy;155 15 140 60 179 32 131 32
             170 60 155 15; lightRed)
  AddPolyline(xyxy;155 15 140 60 179 32 131 32
              170 60 155 15; 1; darkRed)
  AddPolygon(xyxy;102 62 97 77 77 77 92 92
             87 107 102 97 117 107 112 92
             126 77 107 77 102 62;
             gray)
  AddPolygon(xyxy;100 60 95 75 75 75 90 90
             85 105 100 95 115 105 110 90
             124 75 105 75 100 60;
             lightBlue)
  AddPolyline(xyxy;100 60 95 75 75 75 90 90
              85 105 100 95 115 105 110 90
              124 75 105 75 100 60;
              1; darkBlue)
CloseDrawing()
```



**AddSmoothPolyline(scanDirection;listOfCoords;lineWidth;  
lineColor;linePattern)**

The AddSmoothPolyline() function makes it possible to draw smooth lines whose line width, color and pattern can be varied. The arguments *scanDirection* and *listOfCoords* are explained in combination with the AddPolyline() function. Example:

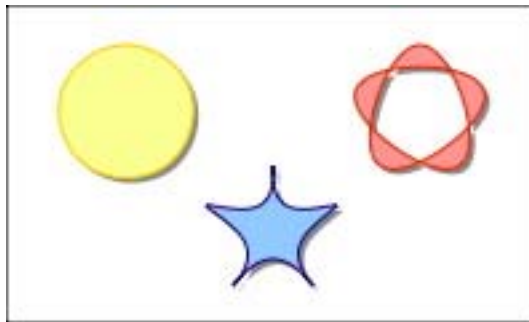
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddSmoothPolyline(xyxy;45 15 69 32 60 60 30 60
                    21 32 45 15;1)
  AddSmoothPolyline(xyxy;155 15 140 60 179 32 131 32
                    170 60 155 15;2;120 120 120)
  AddSmoothPolyline(xyxy;100 60 95 75 75 75 90 90
                    85 105 100 95 115 105 110 90
                    124 75 105 75 100 60;
                    3;200 200 200)
CloseDrawing()
```



**AddSmoothPolygon(scanDirection;listOfCoords;fillColor;  
fillPattern)**

The AddSmoothPolygon() function makes it possible to draw areas with smooth borders whose color and pattern can be varied. The arguments *scanDirection* and *listOfCoords* are explained in combination with the AddPolyline() function. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddSmoothPolygon(xyxy;47 17 71 34 62 62 32 62
                    23 34 47 17; gray)
  AddSmoothPolygon(xyxy;45 15 69 32 60 60 30 60
                    21 32 45 15; lightYellow)
  AddSmoothPolyline(xyxy;45 15 69 32 60 60 30 60
                    21 32 45 15; 1;darkYellow)
  AddSmoothPolygon(xyxy;157 17 142 62 181 34 133 34
                    172 62 157 17; gray)
  AddSmoothPolygon(xyxy;155 15 140 60 179 32 131 32
                    170 60 155 15; lightRed)
  AddSmoothPolyline(xyxy;155 15 140 60 179 32 131 32
                    170 60 155 15; 1;darkRed)
  AddSmoothPolygon(xyxy;102 62 97 77 77 77 92 92
                    87 107 102 97 117 107 112 92
                    126 77 107 77 102 62; gray)
  AddSmoothPolygon(xyxy;100 60 95 75 75 75 90 90
                    85 105 100 95 115 105 110 90
                    124 75 105 75 100 60;
                    lightBlue)
  AddSmoothPolyline(xyxy;100 60 95 75 75 75 90 90
                    85 105 100 95 115 105 110 90
                    124 75 105 75 100 60;
                    1;darkBlue)
CloseDrawing()
```



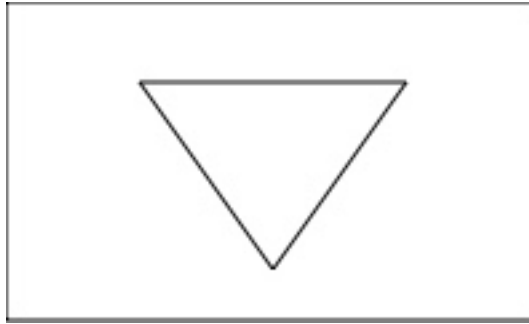
**AddPath(pathData;fillColor;fillPattern;borderWidth;  
borderColor;borderPattern;shadowOffset;  
shadowColor;shadowPattern)**

The function AddPath() is a very general and, at the same time, extremely powerful function for drawing 2D objects. By using the 1st argument *pathData*, the shape of the object is clearly defined. A total of six path constants (1...6) are available for this.

<i>path constant</i>	<i>values</i>
1...close path	(none)
2...move to	xPt yPt e.g.: 2 100 100
3...line to	xPt yPt e.g.: 3 100 100
4...quad. Bézier to	xControlPt yControlPt xEndPt yEndPt e.g.: 4 50 20 100 100
5...cubic Bézier to	xControlPt1 yControlPt1 xControlPt2 yControlPt2 xEndPt yEndPt e.g.: 5 25 20 75 50 100 100
6...elliptical arc to	xEndPt yEndPt xRadius yRadius x-axis rotation (clockwise) [deg] large arc flag: 0...arc ≤ 180 [deg] 1...arc > 180 [deg] sweep flag: 0...counterclockwise 1...clockwise e.g.: 6 100 100 50 50 0 0 1

Path constants and their values are to be separated by spaces, tabs or line feeds. Examples:

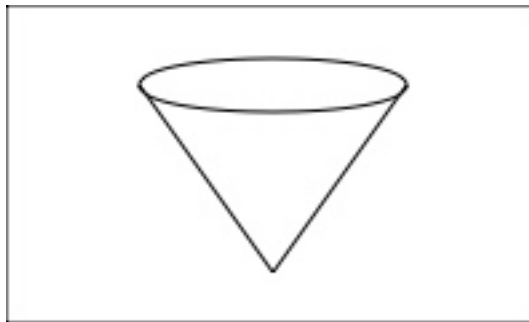
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddPath(2 50 30 // move to 50 30
          3 150 30 // line to 150 30
          3 100 100 // line to 100 100
          1)      // close path
CloseDrawing()
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddPath(2  50  30           // move to 50 30
          6 150  30 50 10 0 0 1 // arc to 150 30
          6  50  30 50 10 0 0 1 // arc to  50 30
          3 100 100           // line to 100 100
          3 150  30)         // line to 150 30
CloseDrawing()

```

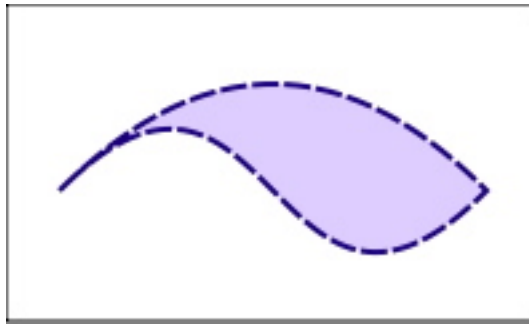


The arguments *fillColor*, *fillPattern*, *borderWidth*, *borderColor*, *borderPattern*, *shadowOffset*, *shadowColor* and *shadowPattern* make it possible to draw objects with fill, border and shadow style if desired. Examples:

```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddPath(2  20  70           // move to 20 70
          4 100 -10 180  70    // quad Bézier 180 70
          5 100 150 100 -10 20 70; // cubic Bézier 20 70
          50 0 255 50;;        // fill
          2 10 2;darkBlue)     // border
CloseDrawing()

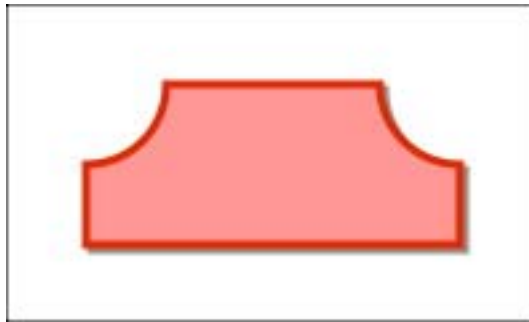
```



```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddPath(2 30 90           // move to 30 90
          3 30 60           // line to 30 60
          6 60 30 30 30 0 0 0 // arc to 60 30
          3 140 30          // line to 140 30
          6 170 60 30 30 0 0 0 // arc to 170 60
          3 170 90          // line to 170 90
          1;                // close path
          255 155 155;;     // fill
          3;darkRed;;       // border
          2;150 150 150)    // shadow
CloseDrawing()

```



A path definition can also be based on several partial paths, i.e. the argument *pathData* contains several closed paths. This makes it possible to define objects with holes.

Example:

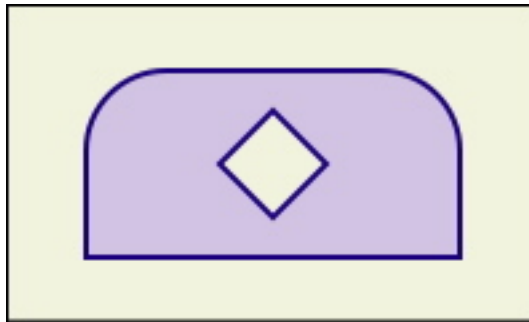
```

OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddPath(2 30 95          // move to 30 95
          3 30 55          // line to 30 55
          6 60 25 30 30 0 0 1 // arc to 60 25
          3 140 25         // line to 140 25
          6 170 55 30 30 0 0 1 // arc to 170 55
          3 170 95         // line to 170 95
          1                // close outer path

          // punch out diamond
          2 100 40         // move to 100 40
          3 120 60         // line to 120 60
          3 100 80         // line to 100 80
          3 80 60          // line to 80 60
          1;              // close inner path

          50 0 255 50;;    // fill
          2;darkBlue)      // border
  Background(242 242 222)
CloseDrawing()

```



**AddSymbol(hCenter;vCenter;symbolType;symbolSize;  
lineWidth;symbolColor;symbolPattern)**

The AddSymbol() function makes it possible to draw symbols whose size, line width, color and pattern can be varied. The arguments *hCenter* and *vCenter* define the center of the symbol. The appearance of the symbol is controlled by using the arguments *symbolType*, *symbolSize*, *lineWidth*, *symbolColor* and *symbolPattern*. A total of 18 symbols are presently available. An overview of the predefined symbols can be found in *xmReference*. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)

  // 1st row
  AddSymbol( 7;15; 1;4)
  AddSymbol( 24;15; 2;4)
  AddSymbol( 41;15; 3;4)
  AddSymbol( 58;15; 4;4)
  AddSymbol( 75;15; 5;4)
  AddSymbol( 92;15; 6;4)
  AddSymbol(109;15; 7;4)
  AddSymbol(126;15; 8;4)
  AddSymbol(143;15; 9;4)
  AddSymbol(160;15;10;4)
  AddSymbol(177;15;11;4)
  AddSymbol(193;15;12;4)

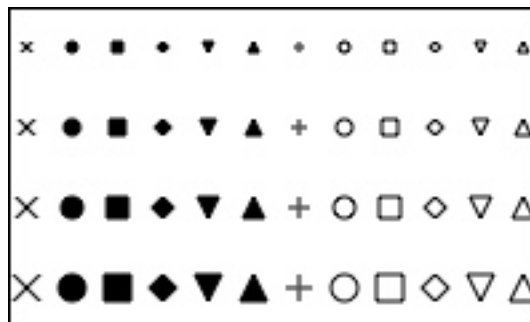
  // 2nd row
  AddSymbol( 7;45; 1;6)
  AddSymbol( 24;45; 2;6)
  AddSymbol( 41;45; 3;6)
  AddSymbol( 58;45; 4;6)
  AddSymbol( 75;45; 5;6)
  AddSymbol( 92;45; 6;6)
  AddSymbol(109;45; 7;6)
  AddSymbol(126;45; 8;6)
  AddSymbol(143;45; 9;6)
  AddSymbol(160;45;10;6)
  AddSymbol(177;45;11;6)
  AddSymbol(193;45;12;6)
```



```
// 3rd row
AddSymbol( 7;75; 1;8)
AddSymbol( 24;75; 2;8)
AddSymbol( 41;75; 3;8)
AddSymbol( 58;75; 4;8)
AddSymbol( 75;75; 5;8)
AddSymbol( 92;75; 6;8)
AddSymbol(109;75; 7;8)
AddSymbol(126;75; 8;8)
AddSymbol(143;75; 9;8)
AddSymbol(160;75;10;8)
AddSymbol(177;75;11;8)
AddSymbol(193;75;12;8)

// 4th row
AddSymbol( 7;105; 1;10)
AddSymbol( 24;105; 2;10)
AddSymbol( 41;105; 3;10)
AddSymbol( 58;105; 4;10)
AddSymbol( 75;105; 5;10)
AddSymbol( 92;105; 6;10)
AddSymbol(109;105; 7;10)
AddSymbol(126;105; 8;10)
AddSymbol(143;105; 9;10)
AddSymbol(160;105;10;10)
AddSymbol(177;105;11;10)
AddSymbol(193;105;12;10)
```

```
CloseDrawing()
```



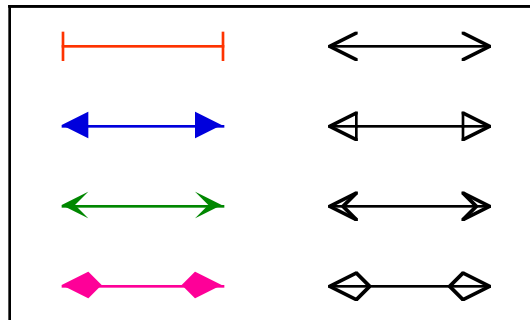
```
AddArrow(hStart;vStart;hEnd;vEnd;lineWidth;lineColor;  
         linePattern;headLocation;headLength;headWidth;  
         headInset;hasHollowHead)
```

The `AddArrow()` function makes it possible to draw arrows whose line width, color, pattern and heads can be varied. Moreover, by using the argument *headLocation*, the arrowhead can be positioned on the end (default), at the beginning or on both of them. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)

  AddArrow( 20; 15; 80; 15;1;red;;begin+end;0;10;0)
  AddArrow(120; 15;180; 15;1;;;begin+end;10;10;10)
  AddArrow( 20; 45; 80; 45;1;blue;;begin+end;10;10)
  AddArrow(120; 45;180; 45;1;;;begin+end;10;10;0;on)
  AddArrow( 20; 75; 80; 75;1;green;;begin+end;10;10;5)
  AddArrow(120; 75;180; 75;1;;;begin+end;10;10;5;on)
  AddArrow( 20;105; 80;105;1;purple;;begin+end;10;10;-5)
  AddArrow(120;105;180;105;1;;;begin+end;10;10;-5;on)

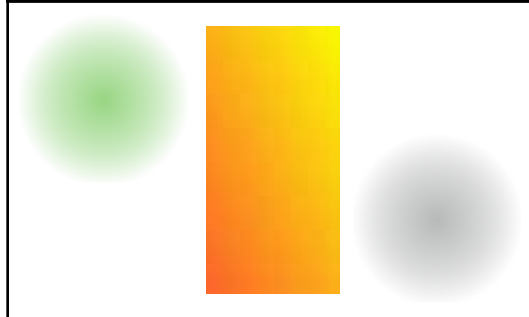
CloseDrawing()
```



**AddPicture(left;top;width;height;sourceType;sourceName;  
location;adjustment;isProportional)**

By using the AddPicture() function, it is possible to add a picture, e.g. a company logo, to a drawing. Three different picture sources are supported for this: the clipboard (*sourceType=clipboard*), a file (*sourceType=file*) or a resource (*sourceType=resource*). The arguments *sourceType* and *sourceName* are explained in detail in the PictureStyle() function in the *Styles* section. The upper left-hand corner of the picture is positioned by using the arguments *left* and *top*. The coordinate origin is located in the upper left-hand corner of the drawing or the view. The arguments *width* and *height* serve to scale the picture. If the width and height are not defined, the actual width and height of the picture are used. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddPicture(75;10;50;100;resource;"32")
  AddPicture( 5; 5;;;resource;"37")
  AddPicture(130;50;;;resource;"33")
CloseDrawing()
```



The arguments *location*, *adjustment* and *isProportional* are explained in the *Background* section, BackgroundPict() function.

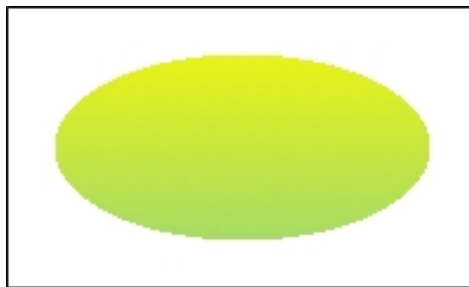
The visibility of the primitives can be limited to a certain area (clipping area) by entering so-called clipping functions.

Currently the clipping functions are supported only in Windows OS. The following clipping functions can be used to define the clipping area:

```
AddClipRect(type;left;top;width;height)
AddClipRoundRect(type;left;top;width;height;hCurvature;
                   vCurvature)
AddClipOval(type;left;top;width;height)
AddClipSlice(type;left;top;width;height;startAngle;
               arcAngle;innerRadius)
AddClipPolygon(type;scanDirection;listOfCoords)
AddClipSmoothPolygon(type;scanDirection;listOfCoords)
```

The setup of the clipping functions and the meaning of the arguments are identical to the primitives. The argument *type* makes it possible to combine several clipping functions. As the default, each clipping function is clipped to the current clipping area (*type=sect*); i.e. the new clipping area can never be larger than the previous clipping area. Example:

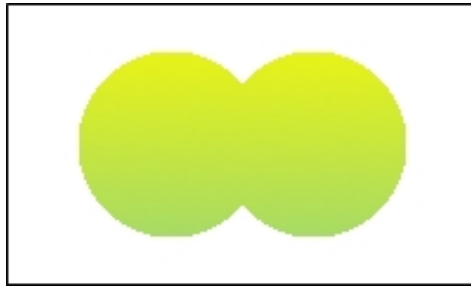
```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddClipOval(sect;20;20;160;80)
  AddPicture(0;0;200;120;resource;"7")
CloseDrawing()
```



Clipping areas are combined if *type=union* is set; i.e. the current clipping area is expanded by the area defined by the clipping function.

Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddClipOval(sect;30;20;80;80)
  AddClipOval(union;90;20;80;80)
  AddPicture(0;0;200;120;resource;"7")
CloseDrawing()
```



If *type=diff* is set, the new clipping area results from the difference between the current clipping area and the area defined by the clipping function. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  AddClipOval(sect;60;20;80;80)
  AddClipOval(diff;90;20;80;80)
  AddPicture(0;0;200;120;resource;"7")
CloseDrawing()
```



### **AddClipReset()**

The `AddClipReset()` function either resets the clipping area to the width and height of the drawing or, if the `AddClipReset()` function is entered within a view, the area is reset to the width and height of the view.

## Output

Output functions make it possible to save a drawing in a file. The following formats are presently available:

- Mac OS X: PDF, PICT, JPEG, PNG, BMP, SVG, TIFF
- Windows: PDF, EMF, GIF, JPEG, PNG, BMP, SVG, TIFF

PDF, SVG and EMF format are vector formats with high print quality, unlike GIF, JPG, PNG, BMP and TIFF formats. The latter represents pure bitmap formats with low resolution (e.g. 72dpi).

If no output function is entered, then, as the default, the drawing is copied into a FileMaker Pro container field. Example:

```
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 2 1 5 3)
  BarChart()
CloseDrawing()
```

All output functions can be entered in any position, both inside and outside of the `OpenDrawing()` and `CloseDrawing()` functions.

### **SendToClipboard()**

The `SendToClipboard()` function contains no arguments and has to be entered when a drawing is to be copied to the clipboard.

Example 1:

```
// The drawing is copied into a
// FileMaker container field and
// copied to the clipboard.
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 2 1 5 3)
  BarChart()
CloseDrawing()
SendToClipboard()
```

Example 2:

```
// The drawing is copied into a
// FileMaker container field,
// copied to the clipboard and
// saved as a PNG file.
OpenDrawing(200;120)
  AddFrame(0;0;200;120)
  ChartData(3 2 1 5 3)
  BarChart()
CloseDrawing()
SendToClipboard()
SaveAsPNGFile("Drawing.png")
```

### **SaveAsPDFFile(fileName;fileFlag;creatorType)**

The `SaveAsPDFFile()` function makes it possible to save a drawing in a PDF file. By using the 1st argument *fileName* the name of the file including an optional file path is defined. The separator in the file path is a slash "/", with no space before and after the slash. For example:

```
SaveAsPDFFile("Macintosh HD/Pictures/Drawing.pdf")
```

Either a complete, absolute file path or only a relative path can be passed to `SaveAsPDFFile()`. The relative path refers to the folder in which the current FileMaker Pro database file is located. If a file with the same name already exists, the following options are available using the 2nd argument *fileFlag*:

- *fileFlag=addCounter*: (default)  
As the default, a counter is added to the file name, e.g. Drawing-1.pdf, Drawing-2.pdf, etc.
- *fileFlag=replace*:  
The file that already exists is overwritten.
- *fileFlag=throwError*:  
Abortion with the error message "File already exists".

Using the 3rd argument *creatorType*, the file creator can be defined in Mac OS X. In doing so, the default application is determined with which the file should be opened. As the default, no creator type (*creatorType*="????") is defined. In Windows the argument *creatorType* is ignored.

Examples:

```
SaveAsPDFFile("Diagram.pdf")
SaveAsPDFFile("Diagrams/Chart.pdf";replace)
SaveAsPDFFile("C:/Programs/Plots/Plot1.pdf";replace)
SaveAsPDFFile("Macintosh HD/Diagrams/Plot.pdf")
SaveAsPDFFile("Plot.pdf";;"8BIM") // save as
// Photoshop-file.
```

**SaveAsEMFFile(fileName;fileFlag)**

The SaveAsEMFFile() function is only available in Windows and makes it possible to save a drawing in an EMF file (enhanced metafile format). In Mac OS X the function SaveAsEMFFile() is ignored. By using the 1st argument *fileName*, the name of the file including an optional file path is defined. The separator in the file path is a slash "/", with no space before and after the slash. For example:

```
SaveAsEMFFile("C:/Programs/Plots/Plot-1.emf")
```

Either a complete, absolute file path or only a relative path can be passed to SaveAsEMFFile(). The relative path refers to the folder in which the current FileMaker Pro database file is located. If a file with the same name already exists, the following options are available using the 2nd argument *fileFlag*:

- *fileFlag=addCounter*: (default)  
As the default, a counter is added to the file name, e.g. Drawing-1.emf, Drawing-2.emf, etc.
- *fileFlag=replace*:  
The file that already exists is overwritten.
- *fileFlag=throwError*:  
Abortion with the error message "File already exists".

Examples:

```
SaveAsEMFFile("Diagram.emf")  
SaveAsEMFFile("Diagrams/Chart.emf";replace)  
SaveAsEMFFile("C:/Programs/Plots/Plot1.emf")
```

**SaveAsPICTFile(fileName;fileFlag;creatorType)**

The SaveAsPICTFile() function is only available in Mac OS X and makes it possible to save a drawing in a PICT file. In Windows the SaveAsPICTFile() function is ignored. The SaveAsPICTFile() function is set up exactly the same as the SaveAsPDFFile() function.

Examples:

```
SaveAsPICTFile("Diagram.pct")  
SaveAsPICTFile("Diagrams/Chart.pct";replace)  
SaveAsPICTFile("Macintosh HD/Diagrams/Chart.pct")  
SaveAsPICTFile("Plot.pct";;"8BIM") // save as  
// Photoshop-file.
```



**SaveAsGIFFile(fileName;fileFlag)**

The SaveAsGIFFile() function is presently only available in Windows and is set up exactly the same as the SaveAsEMFFile() function.

Examples:

```
SaveAsGIFFile("Diagram.gif")
SaveAsGIFFile("Diagrams/Chart.gif";replace)
SaveAsGIFFile("C:/Programs/Plots/Plot1.gif")
```

**SaveAsJPGFile(fileName;fileFlag;creatorType;compression)**

The SaveAsJPGFile() function is available in both Mac OS X and Windows and is set up exactly the same as the SaveAsPDFFile() function. In addition, by using the 4th argument *compression*, the picture quality of the JPEG file can be controlled. Five constants are available for this:

<i>constant</i>	<i>value</i>
min	1
low	2
normal	3 (default)
high	4
max	5

High picture quality means larger files as a result of lower compression and vice-versa, low picture quality leads to smaller files with higher compression of picture information. Unless otherwise defined, the JPEG file created is saved with *compression=normal*. Examples:

```
SaveAsJPGFile("Chart1.jpg")
SaveAsJPGFile("Diagrams/Diagram.jpg";replace;;high)
SaveAsJPGFile("Macintosh HD/Diagrams/Chart.jpg")
SaveAsJPGFile("C:/Programs/Plots/Plot1.jpg";;;max)
```

**SaveAsPNGFile(fileName;fileFlag;creatorType)**

The SaveAsPNGFile() function is available in both Mac OS X and Windows and is set up exactly the same as the SaveAsPDFFile() function. Examples:

```
SaveAsPNGFile("Diagram.png")
SaveAsPNGFile("Diagrams/Chart.png";replace)
SaveAsPNGFile("Macintosh HD/Diagrams/Chart.png")
SaveAsPNGFile("C:/Programs/Plots/Plot1.png")
```

**SaveAsBMPFile(fileName;fileFlag;creatorType)**

The SaveAsBMPFile() function is available in both Mac OS X and Windows and is set up exactly the same as the SaveAsPDFFile() function. Examples:

```
SaveAsBMPFile("Diagram.bmp")
SaveAsBMPFile("Diagrams/Chart.bmp";replace)
SaveAsBMPFile("Macintosh HD/Diagrams/Chart.bmp")
SaveAsBMPFile("C:/Programs/Plots/Plot1.bmp")
```

**SaveAsSVGFile(fileName;fileFlag;creatorType)**

The SaveAsSVGFile() function is available in both Mac OS X and Windows and is set up exactly the same as the SaveAsPDFFile() function. Examples:

```
SaveAsSVGFile("Diagram.svg")
SaveAsSVGFile("Diagrams/Chart.svg";replace)
SaveAsSVGFile("Macintosh HD/Diagrams/Chart.svg")
SaveAsSVGFile("C:/Programs/Plots/Plot1.svg")
```

**SaveAsTIFFFile(fileName;fileFlag;creatorType)**

The SaveAsTIFFFile() function is available in both Mac OS X and Windows and is set up exactly the same as the SaveAsPDFFile() function. Examples:

```
SaveAsTIFFFile("Diagram.tif")
SaveAsTIFFFile("Diagrams/Chart.tif";replace)
SaveAsTIFFFile("Macintosh HD/Diagrams/Chart.tif")
SaveAsTIFFFile("C:/Programs/Plots/Plot1.tif")
```

## Miscellaneous

### **DateTimeOptions(dateOrder;startingDay)**

The first argument *dateOrder* can be used to define the order of day, month and year when entering the date. The function is to be placed before `ChartData()` or before `ChartDataRead()`. For example:

```
OpenDrawing(...)
  DateTimeOptions(...)
  OpenChart(...)
    ... // diagram 1
  CloseChart()
  OpenChart(...)
    ... // diagram 2
  CloseChart()
CloseDrawing()
```

More information and examples can be found in the *Data* section, *Entry of Date and Time* chapter.

The second argument *startingDay* can be used to specify the beginning of the calendar week using a value between 1 and 7, i.e. 1=Sunday (default), 2=Monday, 3=Tuesday, etc.

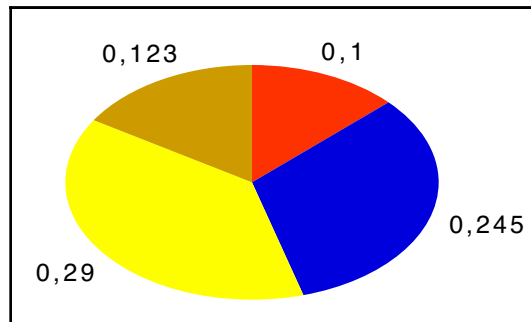
#### Examples:

```
DateTimeOptions(mdy;1) // American date format
DateTimeOptions(;2) // week starts on Monday (ISO 8601)
```

**SetDecimalPoint(char)**

The decimal point symbol can be controlled by using the `SetDecimalPoint()` function. `SetDecimalPoint()` is a "global" function, i.e. all decimal numbers displayed in a chart are concerned, regardless of whether they have to do with axis labels, chart values, etc. and also regardless of possible format specifiers.

```
OpenDrawing(200;120)
  SetDecimalPoint(",")
  AddFrame(0;0;200;120)
  ChartData(0.1 0.245 0,29 0,123)
  PieChart(label+oval)
  BorderStyle(all;none)
CloseDrawing()
```



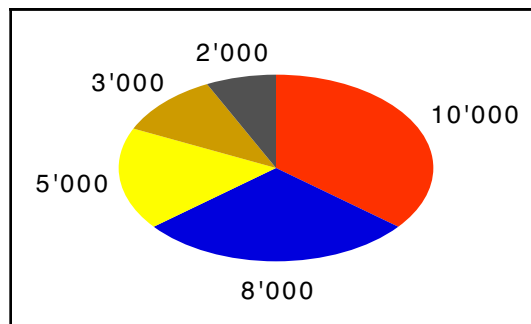
Entering decimal numbers is not affected by the `SetDecimalPoint()` function; both "." and "," are automatically recognized as a decimal point when entering decimal numbers via `ChartData()`.

**SetThousandsSep(char)**

The `SetThousandsSep()` function makes it possible to output numbers with thousands separators. `SetThousandsSep()` is a "global" function, i.e. all numbers displayed in a chart are concerned, regardless of whether they have to do with axis labels, chart values, etc. and also regardless of possible format specifiers.

Example:

```
OpenDrawing(200;120)
  SetThousandsSep(" ")
  AddFrame(0;0;200;120)
  ChartData(10000 8000 5000 3000 2000)
  PieChart(label+oval)
  BorderStyle(all;none)
CloseDrawing()
```



Thousands separators are not permitted when entering numbers; see the `ChartData()` and `ChartDataRead()` functions in the *Data* section.

# Index

# Index

## A

AddArc	294
AddArrow	306
AddClipOval	308
AddClipPolygon	308
AddClipRect	308
AddClipReset	309
AddClipRoundRect	308
AddClipSlice	308
AddClipSmoothPolygon	308
AddEllipse	292
AddFrame	288
AddLine	287
AddOval	293
AddPath	300
AddPicture	307
AddPolygon	297
AddPolyline	296
AddRect	289
AddRoundFrame	290
AddRoundRect	291
AddSlice	295
AddSmoothPolygon	299
AddSmoothPolyline	298
AddSymbol	304
AddText	284
AreaChart	46
AreaChart2D	53
AreaChartOptions	50
ArrowStyle	148
Axes	195
AxisLabelBackground	215
AxisLabelOptions	217
AxisLabelStyle	214
AxisLabelText	211

AxisLine	207
AxisMajorTickLabelBackground	222
AxisMajorTickLabelOptions	223
AxisMajorTickLabelStyle	221
AxisMajorTickLabelTexts	219
AxisMajorTicks	207
AxisMinorTickLabelBackground	226
AxisMinorTickLabelOptions	226
AxisMinorTickLabelStyle	226
AxisMinorTickLabelTexts	226
AxisMinorTicks	207
AxisOptions	209
<b>B</b>	
Background	171
BackgroundPict	175
BarChart	57
BarChart2D	67
BarChartOptions	67
BorderStyle	140
BoxPlot	128
BoxPlotOptions	129
BubbleChart	75
BubbleChart2D	78
BubbleChartOptions	78
<b>C</b>	
CandlestickChart	116
CandlestickChart2D	119
ChartBackground	179
ChartBackgroundPict	181
ChartData	26
ChartDataLowerLimits	29
ChartDataOptions	27
ChartDataRead	31
ChartDataUpperLimits	29
ChartDataWrite	32
Charts	36
CloseChart	25
CloseDrawing	20
CloseView	21
Curve Fitting	267
CurveFitting	268
CurveFittingLineStyle	270
CurveFittingOptions	271



**D**

Data	26
Date and Time	34
DateTimeOptions	315
Drop Lines	247
DropLineReferenceLine	250
DropLineReferencePoint	249
DropLineReferenceSeries	252
DropLineStyle	247

**E**

Error Bars	276
ErrorBarData	278
ErrorBars	276
ErrorBarStyle	279
ErrorBarStyle2D	281

**F**

FileMaker Pro Runtime	9
FileMaker Pro Version	8
FillStyle	136

**G**

GanttChart	70
Graphic Primitives	283
GridFrame	192
GridLocation	193
Grids	184

**H**

HighLowChart	108
HighLowChart2D	115
Histogram	120
HistogramOptions	123
HistogramRange	120

**I**

Installation	9
--------------	---

**L**

LabelBackground	158
LabelOptions	159
LabelStyle	153
LabelTexts	150

Layout	18
Legend	230
LegendBackground	232
LegendOptions	233
LegendStyle	231
LegendTexts	230
LineChart	40
LineChart2D	44
LineStyle	142

## M

MajorGridLineColors	184
MajorGridLinePatterns	184
MajorGridLineWidths	184
MajorGridStripeColors	189
MajorGridStripePatterns	189
MinorGridLineColors	186
MinorGridLinePatterns	186
MinorGridLineWidths	186
MinorGridStripeColors	189
MinorGridStripePatterns	189
Miscellaneous	315
Moving Averages	254
MovingAverage	254
MovingAverageLineStyle	256
MovingAverageOptions	258

## O

OpenChart	21
OpenDrawing	18
OpenView	20
Operating System	8
Output	310
Overview	17

## P

PictureStyle	138
PieChart	80
PieChartAuxLines	88
PieChartCenterLabelBackground	91
PieChartCenterLabelStyle	91
PieChartCenterLabelText	91
PieChartExplodeDepths	87
PieChartExplodes	85
PieChartInnerLabelBackground	90

PieChartInnerLabelStyle	90
PieChartInnerLabelTexts	90
PieChartLabelOptions	89
PolarChart	92
PolarChartOptions	96
 <b>Q</b>	
quartiles	134
 <b>R</b>	
RadarChart	100
RadarChartOptions	104
Requirements	8
 <b>S</b>	
SaveAsBMPFile	314
SaveAsEMFFile	312
SaveAsGIFFile	313
SaveAsJPGFile	313
SaveAsPDFFile	311
SaveAsPICTFile	312
SaveAsPNGFile	313
SaveAsSVGFile	314
SaveAsTIFFFile	314
Scaling	195
ScalingOptions	204
ScatterChart	36
ScatterChart2D	39
SendToClipboard	310
SetDecimalPoint	316
SetThousandsSep	316
ShadowStyle	146
Styles	136
Support	9
SymbolStyle	144
 <b>T</b>	
Title	241
TitleBackground	243
TitleOptions	244
TitleStyle	242
TitleSubStyle	242
TitleText	241