Welcome to
TeeChart Pro version 3
- Activex Control

teeMach
Barcelona

Contents

Introduction

Thank You for using TeeChart Pro ActiveX version 3 - we hope you enjoy working with it!

This manual is aimed at helping those with or without experience of development with TeeChart. There's a Quick Start section to get you up to speed on the fundamentals of TeeChart Pro version 3, a chart type overview and a comprehensive Charting reference section. We've packed in many hints and tips, basing the contents on our experience of types of questions received from TeeChart customers.

1. Introduction

TeeChart Pro ActiveX derives from the 100% Native Delphi product TeeChart Pro version 3 shipped with Borland's Delphi version 3. TeeChart Pro ActiveX version 3 contains many exciting new features.

New for the ActiveX version are direct linking to ODBC version 3 datasources via the Chart Editor and JPEG Chart export. Custom Chart Builder, available with the native Delphi version of TeeChart Pro, is not included in this release of the ActiveX version.

All the underlying native Delphi component architecture has been maintained. Still inherent is the flexibility to mix and match different Series across different charts. New Function definitions can be applied to any group of your data Series across charts. The Chart Editor still offers Series cloning, type change and chart copy. that permits all chart and Series parameters to be copied, as one, to a new chart, thus permiting Copy & Paste inside your programming environment

Series may be connected to ODBC v3 Tables, or SQL Queries. You may use TeeChart Pro ActiveX in Microsoft's Internet Explorer components and deploy them on the WWW.

Licensing issues

Royalty free!! If you wish to distribute your programs compiled with TeeChart components you are free to do so with no royalties!! If you wish to distribute your programs with the possibility that users may access to re-code with TeeChart components therein, or use TeeChart components to re-compile their code you should purchase TeeChart for those users.

Disclaimer

TeeMach SL disclaim all warranties, either express or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to the instructions contained in this manual.

In no event shall TeeMach SL be liable for any damages whatsoever (including without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss), even if TeeMach SL have been advised of the possibility of such damages.

About TeeChart

TeeChart charting components are written in Delphi, by David Berneda. They are available as a 100% Delphi Native Visual Component Library. TeeChart Pro has been interfaced to an ActiveX component by teeMach SL. TeeChart products are distributed and supported by teeMach SL. of Barcelona, Catalonia, Spain.

TeeChart versions

TeeChart version 3 VCL (Version included with Delphi 3)

Runtime TeeChart libraries are included as part of Borland's Delphi version 3. They include many compiled units of the Chart components and all compiled standard Series types. TeeChart v3 has a design time Chart Editor, extensive coded examples and demos, full online help and electronically saved (MS Word format) user manual.

Features of TeeChart 3 runtime version:

- 32-bit version for Delphi 3
- standard Series types
- Statistical functions
- Data aware
- Series Gallery
- 2D, 3D
- QuickReport Integration
- Zoom, scroll and real time
- Royalty free
- Custom drawing
- Custom printing
- Extensive demos
- Design time integrated Chart and Series editor
- Online help
- Electronic reference manual (Word format)

TeeChart Pro v3 VCL

TeeChart Pro is a comprehensive charting tool aimed at those Delphi developers wishing to program with or make use of extended TeeChart functionality. TeeChart Pro includes **100% source code**.

Extended features of TeeChart Pro:

- 100 % Source code
- 16-bit version for Delphi 1
- 32-bit version for Delphi 2 and 3 and C++ Builder
- Extended Series types

- Custom Chart and Series creation
- Developer Custom Series (TeeLoper) guide
- Extended Statistical functions
- Extensive demo code
- Runtime Chart Editor and Gallery
- Extended Online help
- Printed reference manual with developer guide

With **TeeChart Pro** you define the limits. Use the developer guide to help you create hot-spots for drilldown on your charts or customise the TeeChart code to create your own Series types and functions and permanently add them to the TeeChart Gallery. With 100% source code you can choose which functionality to deliver.

TeeChart Pro Activex Control version 3

TeeChart Pro ActiveX version is a comprehensive charting tool for developers across programming language boundaries. It includes extended TeeChart functionality and additional features. TeeChart Pro ActiveX version does not include source code.

Features of TeeChart Pro Activex version:

- 32-bit Activex version for:

・Visual Basic ・Visual C++ ・Delphi ・C++ Builder
・MS Access ・Internet Explorer ・Visual Foxpro
・Crystal Reports

- ODBC version 3 table and SQL query data access
- 11 Standard Series types
- 5 Extended Series types
- Standard and Extended Statistical functions
- Data aware
- Series Gallery
- 2D, 3D
- Zoom, scroll and real time
- Custom drawing
- Custom printing
- Extensive demo code
- Runtime Chart Editor and Gallery
- Online help & HTML format tutorials
- Electronic reference manual (Word format)

With **TeeChart Pro** ActiveX version you may build comprehensive Charting applications across a multitude of programming environments. The extensive demos will show you how to build Charting applications to deploy on the Web or in internal Windows environments.

Getting Started

This section should bring you up to speed if you are trying out TeeChart Pro version 3 for the first time. Building a chart from scratch is very quick with TeeChart so we recommend it worth your while to run through the examples in your own programming environment.

2. Installing TeeChart

TeeChart Pro 3 Activex pack contents

The contents of TeeChart Pro Activex Control doesn't vary for the programming environmment in which you wish to install it. The pack contains code examples and documentation for a variety of programming environments.

TeeChart Components

TeeChart Pro contains components for all the supported operating environments. The correct version specific components are selected at install time. The following sections describe where TeeChart files are installed. No registry entries are modified by the TeeChart installation, Delphi will modify registries at install time but will clean up should you de-install TeeChart from Delphi.

TeeChart components

TeeChart Pro Activex Control contains components for all the supported operating environments. The following sections describe where TeeChart files are installed. No registry entries other than registration of the OCX component are modified by the TeeChart installation.

Contents

OCX and DLL installable files

TeeChart.ocx

Online help

TeeChartx.hlp
TeeChartx.cnt

Tutorials

HTML Format TeeChart Tutorials with HTML help reference.

Manual (MS Word format)

TeeChartx.doc

A printable starter guide and Quick reference to the TeeChart Pro Activex Control. Formatted as Word 95 document.

Example code

There are extensive coded examples included with TeeChart Pro. You will find them in the directory tree below your TeeChart installation directory.

Other files

Please refer to the welcome.txt file, in the directory where you installed TeeChart, for an overview of the files included with this release of TeeChart.

Installation steps

All TeeChart installation files are included in one executable file, **TeeChartx.exe**. Running the file will lead you through the installation steps. At the end of the install the welcome.txt file will display to tell you of any important release information. The files are decompressed and installed in the directory of your choice. Default installation directory is 'TeeChart Pro Activex Control' below your default program installation directory.

The TeeChart installation will automatically register the TeeChart.ocx file.

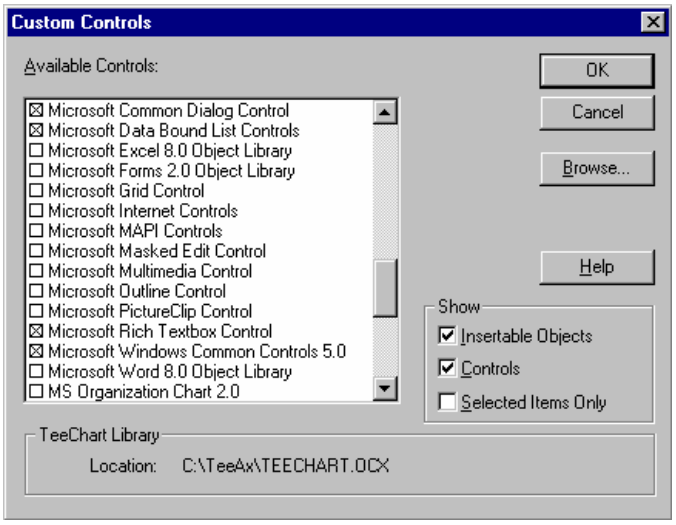
Next you will then need to install the components in your programming environment. You should follow the version specific steps outlined below:

Visual Basic 4

To install TeeChart Pro ActiveX v3 in Visual Basic 4, open the **Tools** menu and select **Custom Controls**.

Fig. 2. -1.

The 'Custom Controls' screen in Visual Basic 4.



Select TeeChart.ocx from the list of available controls the '**OK**'. TeeChart is now installed on your toolbar !

Visual Basic 5

To install TeeChart Pro ActiveX v3 in Visual Basic 5, open the **Project** menu and select **Components**.

When you have selected TeeChart in your components window select **OK** and the TeeChart icon,  , will appear on the VB toolbar.

3. Using TeeChart on-line help and tutorials

Using TeeChart on-line help

Selecting the TeeChart component

Whilst designing your project using the TeeChart component you may select the component from the component palette or on the form where you have placed it and key **F1**. Context sensitive help relating to that component will appear.

Selecting a TeeChart property, method or event

Highlight a property or event in Object Inspector and key **F1** to obtain help for that item. Alternatively write the word for which you would like some help in your code. Select the word by placing and clicking the cursor on that word and key **F1**. TeeChart's context sensitive help will take you directly to the description relating to that functionality.

Help in the TeeChart Chart Editor

At the top, righthand side of the Chart Editor box you will see a button with the "**?**" symbol. Drag that onto any item in the Editor for which you wish to receive help and TeeChart's online help will show you the equivalent runtime functionality of that item.

Accessing the Tutorials

The contents page of the Online helpfile, accessible via the TeeChart Pro v3 Activex Control's icon group of Windows 95 or NT programs menu, contains a link to the first page of the Tutorials. Tutorials are in HTML format and, due to Activex content of some pages, are best seen with an Activex enabled browser such as Internet Explorer.

4. Editing charts

Navigating the Chart Editor

Pages of the Chart Editor

How can we best navigate in the Chart Editor? - We hope you find the Chart Editor reasonably intuitive. For all configuration relating to the chart and not specific to a particular Series got to the **Chart page** of the Chart Editor. For any parameters that apply specifically to a Series go to the **Series page** where you may select the Series from a drop-down combo box. Selecting a Series activates the Series definition tab pages to the parameters of that Series.

Page tab sections of the Chart Editor

Each Page of the Chart Editor has several tabs that contain the configuration parameters of the page. Each tab groups 'like' parameters, for example parameters that apply to axis format are grouped together on the axis tab of the Chart page.

Calling the Chart Editor at runtime

Use the Showeditor method. eg:

```
TChart1.Showeditor;
```

5. A TeeChart quick start guide

The TChart Control

There is one TeeChart icon in the component palette.

TChart

TChart is the basic building blocks of all TeeChart charts.

Creating a new chart with *TChart* component

This section describes the common steps needed to start defining either your chart.

Create a new Form.

Create a new form and place a TChart component on it (Select it on the toolbar and drag it to size on the new form).

Drag the corners of the new TChart component out to a size that helps you visualise the contents of the new chart as you define it. Later you can adjust the size of the chart to suit your needs.

Edit the new chart

Position the mousepointer over the new chart and press the right mousebutton. A menu appears that includes the 'Edit' option. This menu may vary slightly depending on the programming environment in which you are working.

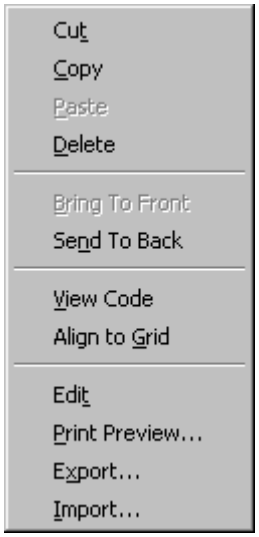


Fig. 5. -1.
Using the right mouse button on the chart will call the Chart options menu.

Select the **Edit** option to edit the chart and define and populate its data Series.

The Chart editor

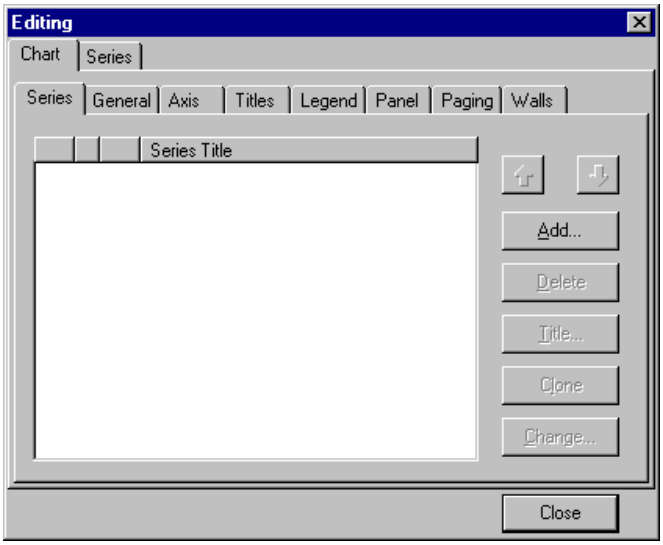


Fig. 5. -2.

The Chart Editor screen

The Chart page (1st page) of the Chart editor contains definition information for the chart. There are various sections to define general and other more specific, chart parameters. Some parameters won't apply until you have some data Series defined in the chart. Try modifying a parameter, the Title for example, and you will see it update in real time on the chart.

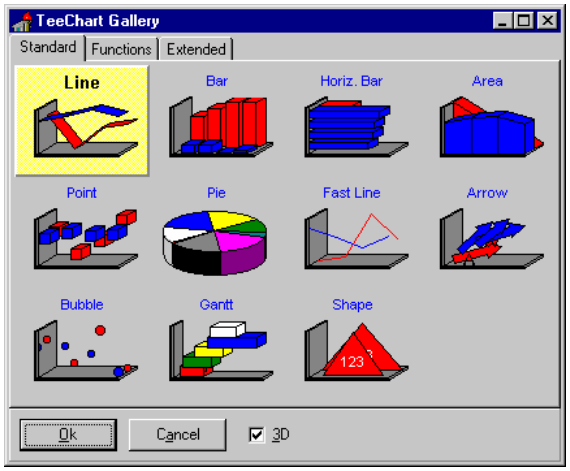
Let's chart some data !! To do that we need to create a data Series.

Add a data Series

Press the **Add** button in the **Series** tab section of the Chart page. TeeChart will show you a gallery of Series types. Select one to add to your chart, you can change its type later if you decide that you would prefer to visualise your data in a different way.

Fig. 5. -3.

The Chart Gallery



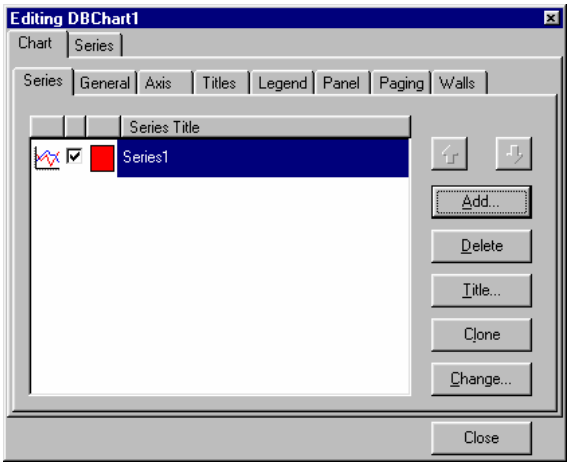
Select a Series type by mouse or with arrow keys and press 'OK'. Double-clicking on the Series type will achieve the same result. The Extended Series page is only available in the Gallery with TeeChart Pro.

The Series type is automatically added to your chart. In the Chart Editor you will see a new configuration tab added for the new Series.

Suppose we had selected a line Series. The editor will appear as follows:

Fig. 5. -4.

New Series added to chart



TeeChart has added the Series to the chart. It has also added some random values so that you can visualise, in the chart, the Series appearance at design-time to easily follow any changes you are making.

Compile your project. The project should compile to show you an empty chart. The random values don't work at runtime so the next step is to go back to the chart editor and add a data source or write your own code to add data values.

Edit the Series

Selecting the Series tab allows you to edit your Series. The next step is to add data to the Series. The steps necessary to include the data for the Series vary slightly depending on whether you are going to program your data input or if you wish to access an ODBC datasource directly. See the following 2 *sections:*

Programmatically adding data to a Series for a description on how to programmatically add data to a TChart Series.

***Configuring an ODBC datasource for a* Series** describes the actions necessary to add an ODBC datasource to a TChart Series.

Programmatically adding data to a Series

We have a TChart on the form and have added a Series to it. We are ready to populate the Series.

Let's type some code to add points values programmatically.

We'll see later how to create a database-aware Chart with automatic record retrieval.

Example Pie-Series

Suppose the Series we added were a Pie Series. We could populate the Series in the following way. For the following code to work we should leave the Series name as its default of *Series1*.

Visual Basic

Place a CommandButton on your Form and go to the OnClick event .
Copy the following code at the Command1.Click event:

```
Private Sub Command1_Click()
  With TChart1.Series(0)
    .Add 40, "Pencil", vbRed
    .Add 60, "Paper", vbBlue
    .Add 30, "Ribbon", vbGreen
  End With
End Sub
```

A description of the Add method and other available methods and properties is available at design time via the context sensitive help included with TeeChart. Place

the cursor on the word *Add* in your code and press **F1** for a full description of the method.

Return to your code and run the project.
Press **Command1** to see the Pie chart appear - the code works !!

Editing the Series

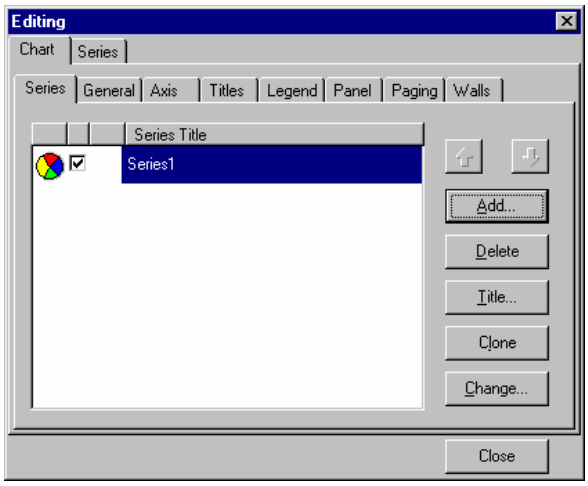
Close the program to go back to design mode.

In design mode, all Chart and Series properties are accessible by **right-clicking** the Chart and selecting the 'Edit' option. Here we'll use the Chart editor to edit our new chart and Series. **Fig. 5** Shows the first editor screen.

To edit features of the Series we can double-click on the Series in the list or highlight the Series and select **Edit** or directly select the tab for the Series - each technique will take us to the editor for the Series.

Fig. 5. -5.

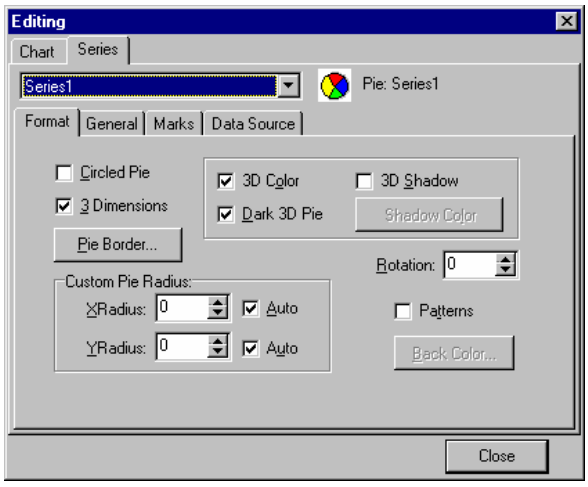
The Chart Editor screen showing the new Pie Series



Try changing some properties of the Pie Series; you will see the results update automatically on the chart. There is no cancel to undo changes but most parameters are easily toggled. Remember, in design time the TChart component hasn't yet run your code so it shows the randomly generated set of data. You will need to start your project to see how the new parameters show up with your own data.

Fig. 5. -6.

The Chart Editor Series page showing editor tab for our Pie Series



Many things can be done visually in the editor, or try modifying some parameters in Object Inspector, or modifying a property with your own code. For more advanced projects, you'll most likely find yourself typing some code.

Now we'll take a closer look at how to add data using an ODBC datasource.

Configuring an ODBC datasource for a Series

Prior to using any database facilities, you should have correctly installed ODBC version 3 in your Windows environment.

TeeChart's ODBC demo database

We have included demo database tables in DBF format as an example ODBC datasource. The ODBC DSN, called **TeeChart Sample Data**, is automatically added to your system when TeeChart is installed.

A Step by step guide to creating a database aware chart....

- Select and install a Pie Series as described in section **'Edit the new chart'**.
- Go to the **Series page** of the **Chart editor** by either double-clicking on the Series name, highlighting the Series and selecting **Edit**, or by selecting the Series tab.
- Select your Series from the Series listbox and go to the datasource tab.
- From the listbox select your data source type.

There are 4 types of data source available. The DataSource style is a component property that may be one of the following:

1. **'No data'** if points are being added **programmatically** (at source code).

or...

2. **'Random values'** draw your chart Series with random values.

or…

3. **'A function'** which may be:

One or more other Series (like LineSeries1, BarSeries2, etc.) which could come from this chart or from another chart in your project. **A *function*** may work with a combination of one or more other Series and an algorithmic function (min, max, average, etc.).

or..

4. An **'ODBC Database'** which may be:

Any valid ODBC v3 Datasource

Adding a dataset

For our example, let's choose the 4th option from the previous section, **'ODBC Database'** style. In our TChart we'll include data from the example MS Access mdb database included as a demo with TeeChart.

You need to select the listbox option an **ODBC database** to map this Series to a table in the teedemo mdb database. As you make the selection you will notice new page options appear to define the dataset for the pie Series.
Your editor should look like **Fig. 5-7**.

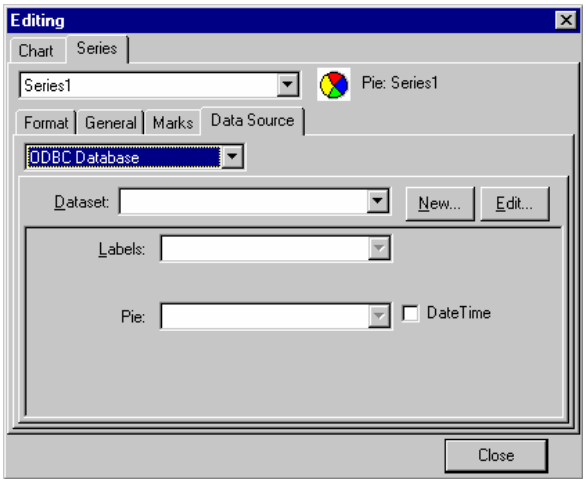


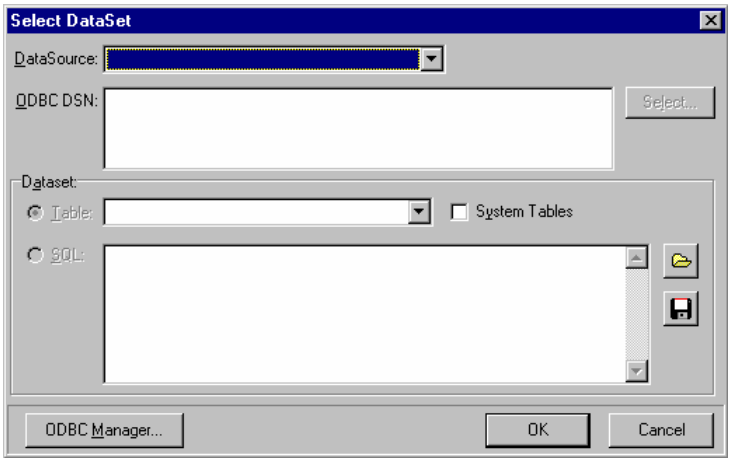
Fig. 5. -7.

The Chart Editor screen showing the Data source tab for our pie Series

You will need to select the **New** button which will open the TeeChart ODBC Connection Window.

Fig. 5. -8.

The TeeChart ODBC Connection Window



1. Click on the down-arrow in the **'Datasource:'** combobox.
This will show a list of all predefined ODBC DSNs.
2. Select your DSN.
3. Highlight the radio button **Table**:
4. Select the table

As we have defined a Pie Series we are presented with pie specific configuration options. Each Series has parameters that vary, but only slightly, in definition from the example below.

The **Pie** field is mandatory, and must contain a **valid numeric** field or a string field with values that can be converted to numbers.

In our example, choose the **Weight** field.

(The *ANIMALS* table is a collection of animal names with weight and size information.)

You can optionally select an **Labels** field to draw the corresponding **string** for each point on the pie.

In our example, choose the **Name** field.
This label will be used to draw the Pie **Legend**.

Note

Some chart Series types have more than one figure to represent a point. Values that are plotted by date or time are an example of **X Values** Series. Not all date-time values should be equally spaced. The **X Value** is the specific **horizontal position** for each **Y value**. Later try deleting the pie Series and adding a Series of a different type, - ie. LineSeries -, you will see the option to apply X and Y values appear.

(We don't have X Values in a Pie Series.)

Now click the **"OK"** button.

The TDBChart component will show a **Pie** Chart and a **Legend**.
You are seeing **exactly** the same chart you saw when you pressed **F9** to run the project.

6. Using the TeeChart demos

You have seen the most basic TeeChart procedures.
The next step is to look at some examples in the demo files.

First Thing To Do:

Open the **Examples** folder below your TeeChart installation Folder.

There are many subfolders containing TeeChart example code for a variety of operating environments.
Each example shows a specific TeeChart feature.

Many Forms contain Source code to show event handling or interactive charting.

In the Visual Basic 4 and 5 folders you will find a folder entitled **Scrolling**. Let's see the **ScrollForm1.Frm** demo form. You may open this with notepad or in Visual Basic by openaing the .vbp project file. ScrollForm1 shows a basic TChart component with one **Line Series** component.

Click on **Form Load**, you will see that it calls **FillDemoPoints**. Switch to source code to see the following:

```
Private Sub Form_Load()
  FillDemoPoints
End Sub

Private Sub FillDemoPoints()
Dim t As Integer

  With TChart1.Series(0)
    'fill the LineSeries with some random data
    .Clear   ' <-- this removes all points from Series1

    'let's add 60 minutes from 12:00 to 12:59
    For t = 0 To 59
      .AddXY TimeValue("12:" & t & ":00"), Rnd(100), "", vbRed
    Next t

   'let's add 60 more minutes from 13:00 to 13:59
    For t = 0 To 59
      .AddXY TimeValue("13:" & t & ":00"), Rnd(100), "", vbRed
    Next t
  End With
End Sub
```

This code will **Clear** Series(0) and plot two hours of **random** values from 0 to 100.

Because it is placed on the **Form Load** event, each time the form is shown, this code is executed.

Double click the **Command1** component to see its associated **Click** code:

```
Private Sub Command1_Click()
Dim h, m, s As Integer
  m = Minute(TChart1.Series(0).XValues.Last)
  h = Hour(TChart1.Series(0).XValues.Last)
  'add a new random point to the Series (one more minute)
  s = 0
  m = m + 1
  If m = 60 Then
    m = 0
    h = h + 1
  End If
  TChart1.Series(0).AddXY (TimeValue(h & ":" & m & ":" & s)), Rnd(100), "",
vbGreen
```

The next source code is the demo purpose:

How to scroll the Horizontal Axis as new Points are added to the end ?

The easy way is to *type* the following code at the continuation in the **Command1.Click** event. This is called *each time* a new point is **added** to the Series.

```
'Now scroll the axis to show the last hour of data
  ScrollHorizAxis

End Sub
```

In this example, we **scale the bottom horizontal chart axis** to show 55 minutes prior to the last point and 5 minutes after the last point.

```
Private Sub ScrollHorizAxis()
  With TChart1.Axis.Bottom  ' <-- with the Horizontal Axis...
      .Automatic = False        ' <-- we dont want automatic scaling

      ' In this example, we will set the Axis Minimum and Maximum
      ' values to show One Hour of data ending at last point Time
      ' plus 5 minutes

      .Maximum = TChart1.Series(0).XValues.Maximum + _
                 TimeValue("0:5:0")
      .Minimum = .Maximum - TimeValue("1:0:0")
  End With
End Sub
```

This code increments the **last added point** Time by one more minute.

The incremented Time value is added to the Series, thus giving a new point.

Many possibilities can be achieved by manually setting the Axis properties.

Each Chart component has four axis, accessible via the TChart **Axis** property: **Left, Top, Right and Bottom**. Each Axis has a boolean "**Automatic**" property that defaults to **True**, meaning TeeChart will **always calculate** the **Minimum** and **Maximum** values for each Axis.

The **Minimum** and **Maximum** properties are of type **double**, allowing float numbers or **DateTime** representations like "Date", "Time", "Now" or any other valid datetime value.

We have seen a specific TeeChart feature in detail.

Now, compile and execute this project *and look at each sample form*.

They should give you ideas that you can apply to your projects while you learn about TeeChart features.

The **TEECHARTX.HLP** file describes each component property in detail, with some source code examples.

We suggest that you to play with each different sample project and create new small projects to test TeeChart with your *real data.*

Charting reference

This section describes design considerations specific to each chart and Series type. Rather than attempting to list all properties that are, anyway, better covered and more easily navigable via the online help file, we would like to bring to your attention points of interest that may help in the design of your application.

To understand the design paradigm of TeeChart we need to separate conceptually, the contents of the chart -the data Series- from the chart itself, e.g. its axis format, legend and titles. We can define and work with Series across chart boundaries, referencing Series components independently of the chart. We may, however, copy and paste a chart and it will copy with all its defined contents, axis, legends and Series.

7. The Chart components

Chart component

There is 1 chart component. TChart.

Chart subcomponents

The Chart component has subcomponents. If, for example, you wish to access or modify the property of an axis of your chart you will be modifying the property of one of the following subcomponents:

```
Axis.Top
Axis.Left
Axis.Bottom
Axis.Right
```

See the online help to get a complete list of axis properties. Legend is another example of of a TChart subcomponent.

The Chart as backdrop to your Series

The Chart components provide the backdrop for your data Series. The overall 'look' of your chart is controlled by the chart properties accessible via the Chart Editor, or by coding - see the online help for a complete list.

As a backdrop you have available to you Canvas property and methods to calculate screen co-ordinates from values and vice-versa.

Associating a Series with a Chart

You may add a Series to a TChart with the Chart Editor or by code.

Via the Chart editor

After a Chart has been placed on a form, a right button click gives access to the Chart Editor via the **Edit** menu option (some programming environments support double-click on the Chart). The first screen of the Edit chart menu offers a list of button options to add, delete, clone, change type or title of a Series. Adding a data Series via this menu automatically associates the new data Series with the chart.

Via code

Use the TChart.AddSeries property to add a Series to a chart.

Example:

In this example we'll create and populate a new Series, Clearing out any Series that already exist in the Chart. You can find this code in the demo 'Add Series' (SeriesProject.vbp):

```
With TChart1
    'Look to see if there's already a Series in the Chart
    If .SeriesCount > 0 Then
    'Remove the existing Series
    .RemoveAllSeries
    End If
    'Add and randomly populate the new Series
    'eg. .AddSeries(scHorizBar) is the same as .AddSeries(2)
    .AddSeries (Combo1.ListIndex) 'Combo Lists assorted Series types
.... 'Fill the new Series with random values
    .Series(0).FillSampleValues 10
    .Series(0).Title = Combo1.List(Combo1.ListIndex) & " Series"
End With
```

Often used parameters

All the parameters associated with charts are useful at the time you need them. There are a few parameters, however, that are worth mentioning here in special context. They are often used or serve to change the overall 'look', performance, or exportability of your chart.

For a more detailed study of these and many other configurable parameters please refer to section Chart reference.

Label increment, % separation and angle

It may be difficult to fit all your axis labels onto the axis that displays onscreen ie, this is often true of date labels which are long and can have a high incidence of change on the axis.

Label Increments

TeeChart offers **Label increment** as a means to controlling the frequency of labelling on your Chart axis. You can access the property via the Chart Editor under **Axis** (select your axis) - **Scales** - **Desired increment**. The menu shows you a list of the standard options for time increments (e.g. One second, one minute … one day, etc.) Selecting 'one day' will cut out repetitions of date on the axis, showing the date label only once for the point range of that date.

Via code you may access the property as a property of the subcomponent for the relevant axis.

Example:

```
TChart1.Axis.Bottom.Increment = TChart1.GetDateTimeStep(dtOneHour)
TChart1.Axis.Right.Increment  = 1000
```

% Separation

If the labels on the axis have a look of rolling into one another you can increase the % separation which will forcibly increase the gap between them. In some cases this may, as a result, hide some labels. It will very much depend on your labels and whether your chart users will have the option to 'resize' their chart.

The option is available in the Chart Editor under **Axis** (select your axis) - **Labels** - **% Separation.**

Setting Labels Separation to 0 % means no overlapping checking will be performed, thus all axis labels will be displayed.

Angle

If it is neither practical or possible to place the axis labels side to side along the axis another option is change the angle of the labels to 90º to the axis. This option is available in the Chart Editor under **Axis** (select your axis) - **Labels** - **% Angle**.

Export File formats (JPEG, BMP) or Metafiles

TeeChart Pro Activex Control offers JPEG, bitmap (BMP) and Metafile format to save charts. Two type of metafile formats are supported WMF (Windows Metafile Format) and EMF (Extended Metafile Format).

Bitmap format is used internally by TeeChart. It is quicker to draw onscreen than a Metafile. When requiring a Chart to be 'exported' to another application or to be embedded in a container application such as MSWord a Metafile format handles resizing better onscreen.

Zoom

TeeChart offers as default zoom on all charts. To obtain zoom at runtime hold the left mouse button and drag mouse toward down/right. You'll see a rectangle around the selected area. Release the left mouse button to Zoom. You can continue zooming again and again. To RESTORE (or UNDO) the zoom, drag a rectangle in the opposite direction (up/left).

Note
You may use mouseclicks to activate code associated with points in data Series. That OnClick behaviour will override zoom behaviour. You can set toggle between OnClick events and zoom with the CancelMouse property.

You may enable or disable zoom functionality in the Chart Editor - Chart page, General tab. The option, AllowZoom accesses the chart property, also changeable by code:

```
TChart1.Zoom.Enable = False
```

'False' deactivates zoom.

Zoom can also be obtained via coding. It is necessary to obtain the screen pixel co-ordinates for the area in which you wish to do the zoom.

Example 1:
Zoom an area with "pixel" co-ordinates:

```
Dim Left, Top, Right, Bottom as integer
Left = 123
Top = 67
Right = 175
Bottom = 100
TChart1.Zoom.ZoomRect Left, Top, Right, Bottom
```

Example 2:
Zoom an area with point value co-ordinates:
You need first to translate from value to pixel co-ordinates. To do so, you can use the Axis or Series components which both share conversion from index value to pixel methods.

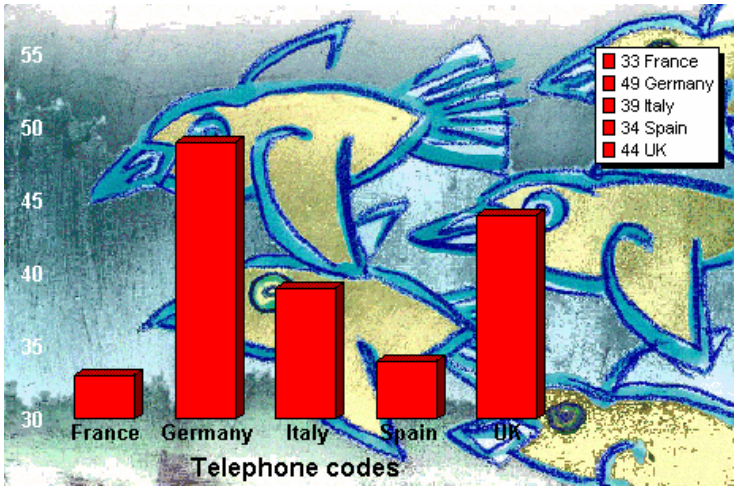
```
Left   = TChart1.Series(0).CalcXPosValue(22.5)
Top    = TChart1.Series(0).CalcYPosValue(5000)
Right  = TChart1.Series(0).CalcXPosValue(57.6)
Bottom = TChart1.Series(0).CalcYPosValue(15000)
TChart1.Zoom. ZoomRect Left, Top, Right, Bottom
```

Backdrop

You can add more information to your chart or simply enhance its appearance by adding a bitmap as a backdrop.

Fig. 7. -1.

Bitmap as a chart backdrop



You will find the parameters to define your Chart backdrop in the **General** tab of the chart page of the Chart editor.

TChart component

The TChart component is the basic building block for all charts. Select the TChart component from the toolbox and simply drag it onto your form to include a chart in your application.

Database connectivity

The Datasource for the Series is defined by the Series definition - not by the chart - Thus multiple Series in a TChart could access different data sources.

Creating the Dataset

TeeChart charts will connect with all ODBC version 3 compatible datasources.

The minimum that requires to be set in each case is:

1. The name of the ODBC DSN,
2. For the table the TableName of a table
   or
   in the case of the query a valid SQL string.

Connecting your Series to a database Dataset

In 'Getting Started' you saw how to connect a data Series to a database dataset. Let's recap here on the key components.

When you select your new **Series** in the **Chart Editor Series page** you will see the tab option for **Data Source**. If you want your data Series to be connected to a new dataset then you should select **ODBC Datasource** from the drop down combo listbox. A new option tab will appear with the options for definition of the new datasource.

The exact contents of the page will change depending on the Series type you have chosen. The parameters you set here modify the properties in the Series definition which vary between Series type. The following table shows the possible options for normal Series types (function Series are different, their datasources are other Series, please refer to the Functions section of this document).

SERIES TYPE	DATASOURCE PROPERTIES
Basic	
Line	XValues, YValues, XLabel
Fast Line	XValues, YValues, XLabel
Bar	XValues, YValues (called Bar), XLabel
Area	XValues, YValues, XLabel
Point	Xvalues, YValues, XLabel
Pie	PieValues, XLabel
Arrow	StartXValues, StartYValues, XLabel, EndXValues, EndYValues
Bubble	Xvalues, YValues, XLabel, RadiusValues
Gantt	StartValues, EndValues, AY (Y axis level), AXLabel (Label optionally shown on Y-axis or as mark)
Shape	X0 (Top), Y0 (Bottom), X1 (Left), Y1 (Right)

Extended	
Candle	OpenValues, CloseValues, HighValues, LowValues, DateValues
Error Bar	XValues, YValues, XLabel, ErrorValues
Polar	XValues, YValues
3D Surface	XValues, YValues, ZValues
Volume	XValues, YValues (VolumeValues), XLabel

Coding your datasource

You may populate your Chart at runtime by coding which Series to add to the chart and defining the fields of those Series.

It assumes you have access to an ODBC table, *Table1* with fields *Name* and *Amount*.

```
TChart1.AddSeries(scLine)
With TChart1.Series(0)
 .DataSource = "DSN=SQLSource; TABLE=Table1"
 .LabelsSource = "Name"
 .YValues.ValueSource = "Amount"
 .CheckDatasource;  'Refreshes data
End with
```

8. Series types

Adding values, accessing values

Series class

The Series class is the ancestor of all Series types. When referencing the properties of the Series type with which you are working check, in addition to the properties listed with that Series, the properties of Series in the online help to get a full list of inherited properties common to all Series.

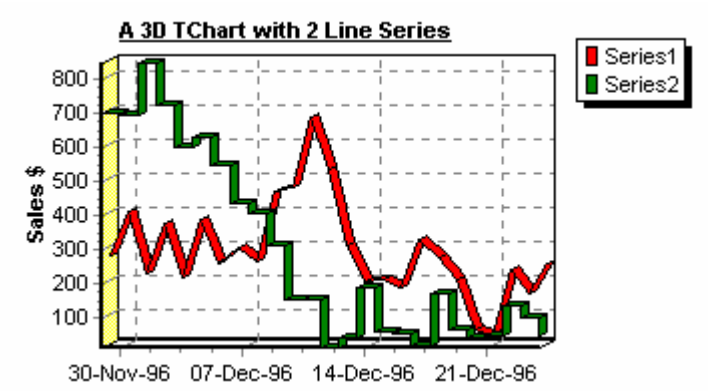
Line Series

There are 2 line Series types available, Line and Fast Line. Which one should be used ? Fast line is just what its name describes - it is fast. It is distinct from Line because to achieve speed - speed to add new points to the Series - the price paid is that it foregoes some properties that the Line Series has. See the section on Fast Line for a description of those differences.

Line

Fig. 8. -1.

A 3D Line Series showing one Series with the **stairs** property set to true. The stairs can be inverted.



Fast line

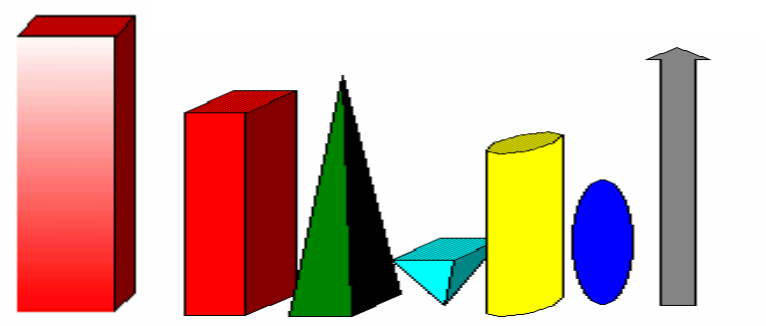
This line Series draws only at 2 Dimensions but draws very quickly - performance will depend on your hardware - The Series type was originally conceived to tackle high volume requirements of technical and financial applications but serves well for any dataset of very high point volumes.

Bar

The bar Series in 2 or 3 dimensions doesn't have to be represented by a rectangular bar -

Fig. 8. -2.

Choose a barstyle for your Chart Series or 'mix and match' to suit your needs.

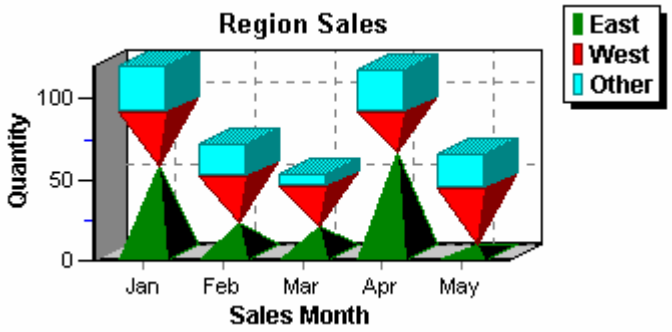


Example bar Series configuration

Mixing bar Series styles may be useful for some applications. Below is an example stacked bar chart.

Fig. 8. -3.

Mixed bartypes



Steps taken to build this demo chart and populate it with data are as follows:

1. Drag a Chart component onto your form and drag it out to the size you want.
2. Click the right mouse button over the chart and select editor from the menu.
3. Add 3 **Bar** Series and give them individual titles using **Title** on the first editor screen (titles in our example 'East' 'West' and 'Other').
4. Select **Multi Bar** and set as **stacked** in the **Format** page of any one of the 3 Series (The change will apply to all 3 Series).
5. Add a title using the entry tab for '**Titles**' in the Chart page of the editor.
6. Go to the axis tab. Select Left axis and add the title. We allowed the axis to automatically size itself but took **minor ticks** off on the bottom axis.
7. Each Series has a different shape defined for its bars. We define the shape individually in the configuration page of each barSeries. The first tab, **Format**, has the option for **Style**.

You could code the style with:

```
With TChart1
  .Series(0).asBar.BarStyle= bsPyramid  'East
  .Series(1).asBar.BarStyle=  bsInvPyramid 'West
  .Series(2).asBar.BarStyle= = bsRectangle 'Other
End With
```

Lookup BarStyle in the TeeChart online help to get the full list of options.

8. For this chart it would be necessary to control the **Series order** to achieve the correct effect. This can be done in the Chart Editor at design time by selecting the Series in the Series list of the Chart page and using the arrows to change the order. At runtime we can achieve the same with the following code:

9. Drawing order is always based on the order for which the Series are assigned to ParenTChart.

The order can be altered at run-time via the ExchangeSeries property:

```
TChart1.ExchangeSeries 0, 1
'Reverses the Z Order between Series(0) and Series(1)
'1 becomes 0 and 0 becomes 1
```

10. We added some random values to populate this demo. We could launch the code at form creation or put a button on the form to run the code.

```
Private Sub Command1_Click()
Dim t As Integer
For t = 1 To 12
  TChart1.Series(0).Add Rnd(70), Format("1," & t & ",1997", "mmm"), vbRed
  TChart1.Series(1).Add Rnd(70), Format("1," & t & ",1997", "mmm"),
vbYellow
  TChart1.Series(2).Add Rnd(30), Format("1," & t & ",1997", "mmm"),
vbGreen
Next t
End Sub
```

11. We populated the data for 12 months but are showing only 5 months. The Paging tab in the Chart page of the editor has the definition of points per page. Set Points per page to 5.

Bar Series display

The **Multi Bar** parameter set in the Format tab of the Series page for one of the bar Series sets the display alignment of the bars for each bar Series in the Chart. - It is not necessary to go to each bar Series to change the parameter.

The following figure shows different formats to display bar Series. Each Chart has the same information only displayed in a different way. The Stacked 100% Series display doesn't represent the actual values but rather the relative value of each element of the Series to a total of 100%.

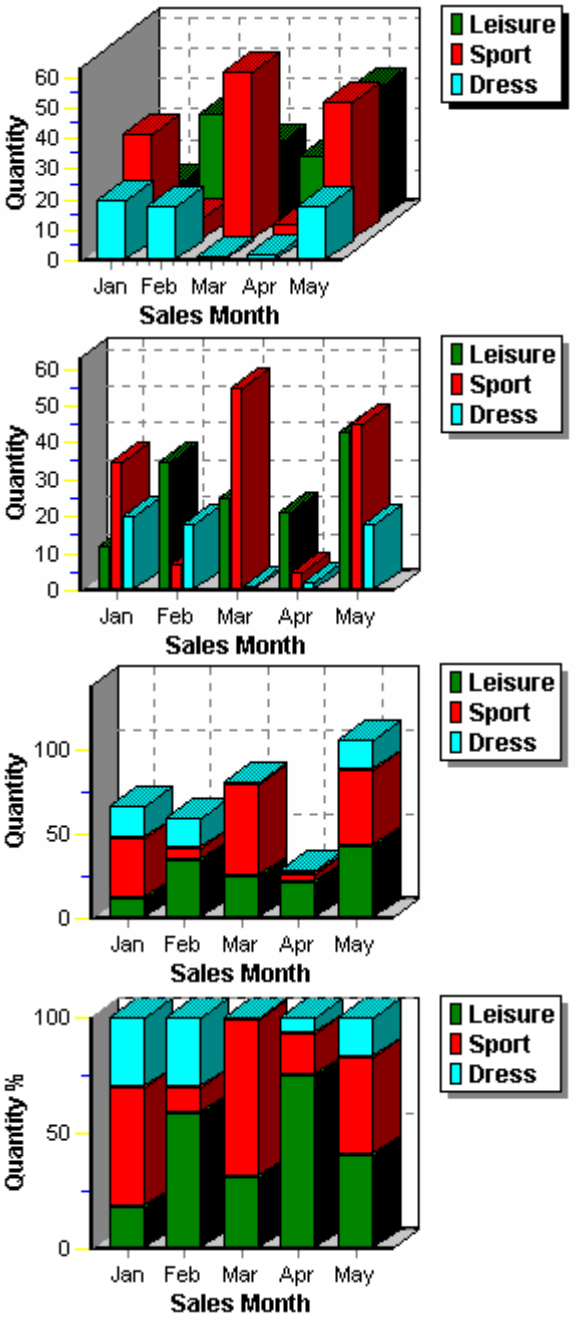


Fig. 8. -4. a

A 3D Bar Series showing four methods to display the same information.

Fig. 8. -4. b

Fig. 8. -4. c

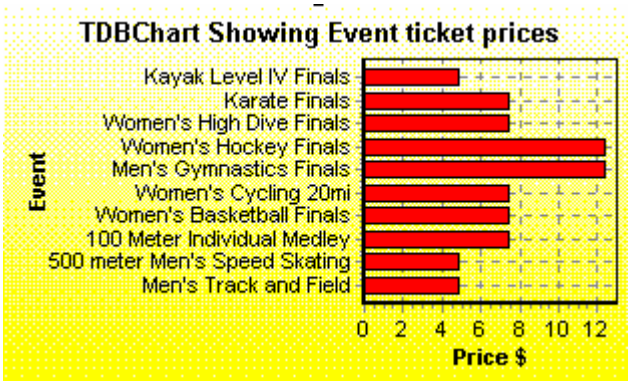
Fig. 8. -4. d

Horizontal bar

The horizontal bar Series shares the same properties as the bar Series. Apart from any aesthetic requirement, one occurrence of the need to use a horizontal bar Series may be to adequately display long axis labels which are best read horizontally.

Fig. 8. -1.

2D Horizontal bar

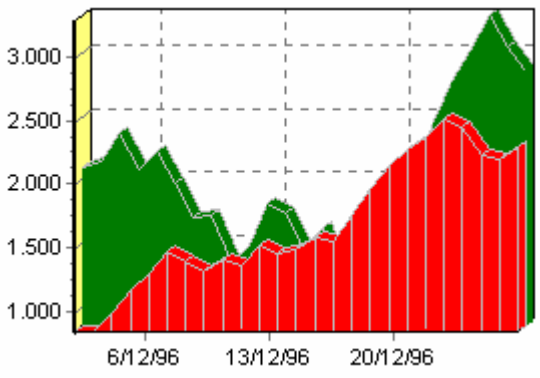


Area

An Area Series has similar characteristics to a line Series - filled -

Fig. 8. -5.

3D Area Series

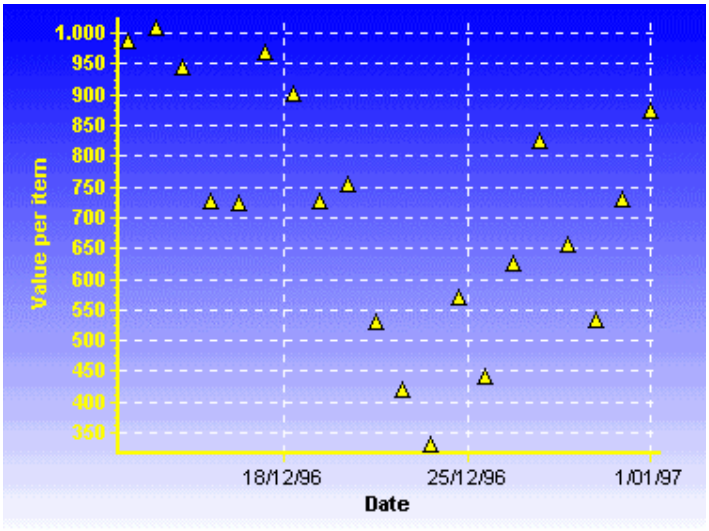


Point

A Point Series is similar in definition to a Line Series without the line.

Fig. 8. -6.

2D Horizontal bar

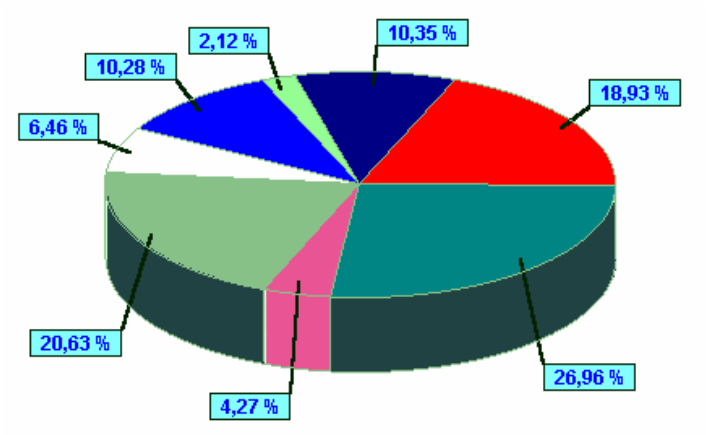


Pie

A Pie Series is unique in not needing any axis. It is possible to mix a Pie Series in a Chart with another Series that requires an axis.

Fig. 8. -7.

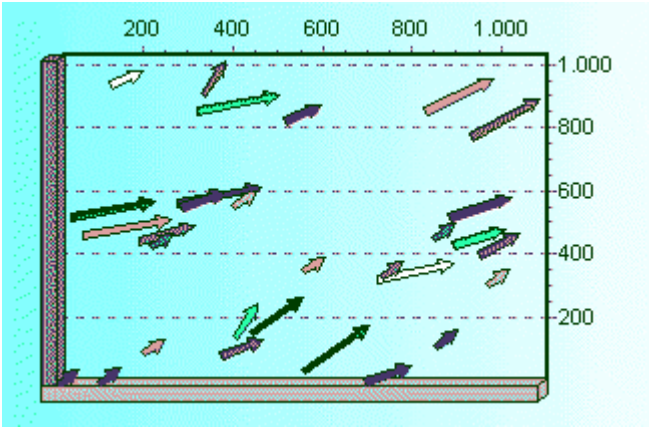
3D Pie



Arrow

Fig. 8. -8.

3D Arrow Series



The arrow Series is useful for displaying start and end points of many individual events.

Bubble

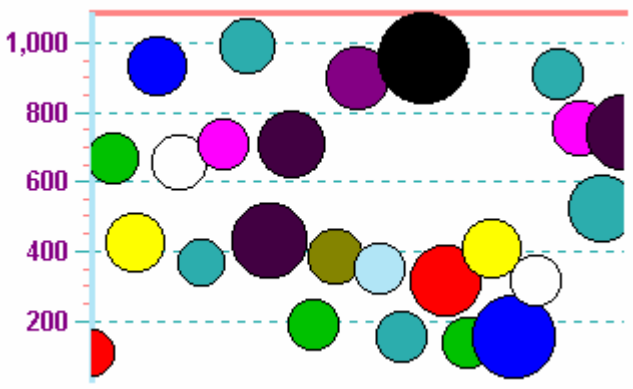
The Bubble Series has 3 configurable parameters that define the value of the data in your Series.

Xvalues, YValues, RadiusValues

The bubble Series is useful for showing importance weighting. For example, comparing high volume selling product that, by income, doesn't bring in a revenue of the scale of another low volume seller. We can see at a glance, literally, that big bubbles are important !

Fig. 8. -9.

2D Bubble Chart



Shapes

Bubble Series can be configured in variable shapes, triangles, etc..

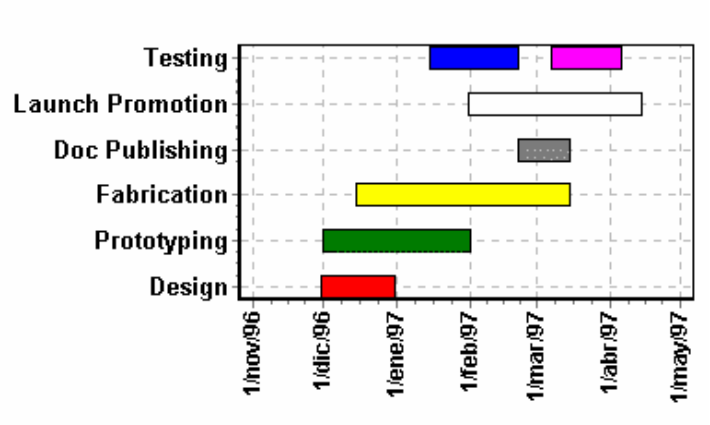
Gantt

Use the Gantt chart as a planner or to track progress of a project or Series of activities.

The Gantt Series draws bars that have start and end values which may be of datetime format. You may define a Y axis value for the vertical position of the bar and you may define 'next bar' to draw connection lines between the bars.

Fig. 8. -10.

2D Chart with a Gantt Series.



How to add Gantt bars manually

Use the AddGantt or AddGanttColor methods.

Example:

```
With TChart1.Series( 0 ).asGantt
   .AddGantt DateValue("1,1,1997"), DateValue("31,1,1997"), 2,_ "Hello"
   .AddGantt DateValue("1997,1,15"), DateValue("1997,2,15"), 1,_ "Nice"
   .AddGantt DateValue("1997, 2, 1"), DateValue("1997, 2, 28"), 0,_ "World"
   .AddGantt DateValue("1997, 3, 1"), DateValue("1997, 3, 31"), 2, ""
   .AddGantt DateValue("1997, 4, 1"), DateValue("1997, 4, 30"), 0, ""
   .AddGantt DateValue("1997, 3, 15"), DateValue("1997, 4, 15"), 1, ""
   'change connecting lines
   .ConnectingPen.Width = 3
   .ConnectingPen.Color = vbBlue
   ' increase bar heights
   .Pointer.VerticalSize = 16
End With
```

Or...

```
With TChart1.Series( 0 ).asGantt
   .AddGantt DateValue("1,1,1997"), DateValue("31,1,1997"), 2,_
"Hello",vbGreen
End with
```

Where "2" is the desired vertical position for this bar.

Choose the vertical position you prefer.

To connect gantt bars:

1) Store the "AddGantt" or "AddGanttColor" function return longint:

```
Dim tmp1, tmp2 as Integer

With TChart1.Series( 0 ).asGantt
tmp1 =.AddGantt DateValue("1,1,1997"), DateValue("31,1,1997"), 2,_
"Development"
tmp2 =.AddGantt DateValue("1997,1,15"), DateValue("1997,2,15"), 1,_
"Testing"
End with
```

2) Then use the NextTask property:

```
TChart1.Series( 0 ).asGantt.NextTask( tmp1 ) = tmp2
```

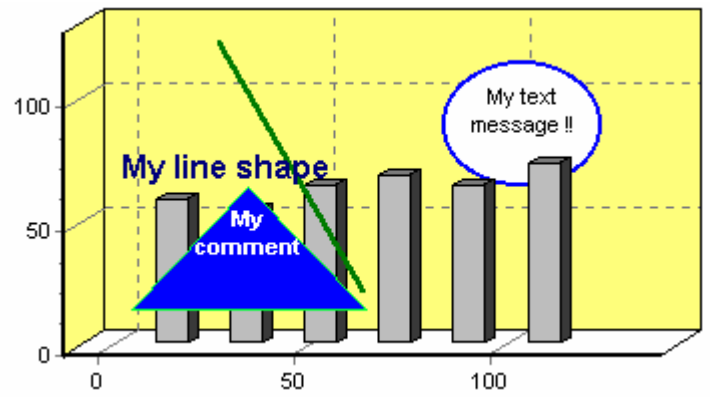
This will draw a line from 'Development' gantt bar to 'Testing' bar. The "ConnectingLinePen" property is the pen used to draw lines.

Shape

Shape Series are useful for defining or adding any additional information to your chart, perhaps in runtime as a result of receipt of exceptional data. You may add text to any shape you add to your chart and relate the shape to another Series.

Fig. 8. -11.

Shape Series



Each shape has two co-ordinates associated with it, top left and bottom right of the invisible rectangle that encloses the shape. You may add text to the box. These co-ordinates and messages could be updated at runtime by your code to dynamically put the shapes anywhere on your chart.

Combining Series

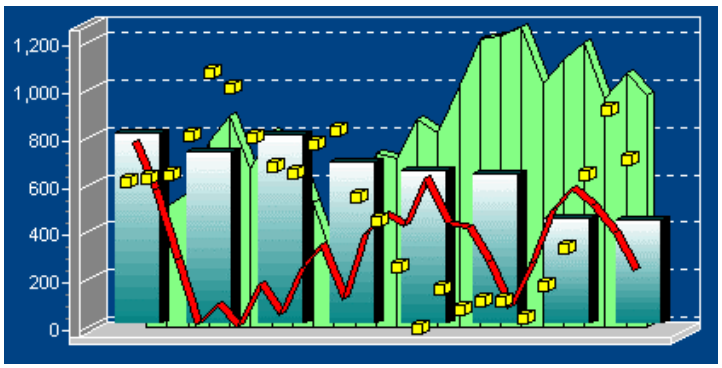
There is no practical limit to the number of Series that you can add concurrently to your chart.

You may mix different Series types, almost any Series type with any other Series type. For certain combinations it is not practical as axis definition between Series may directly conflict. For those cases the Series not available are greyed out in the Series gallery so that you cannot mistakenly select them.

See the section on functions for more about combining Series types. Functions work with other Series to create and display algorithmic relationships.

Fig. 8. -12.

Combining Series

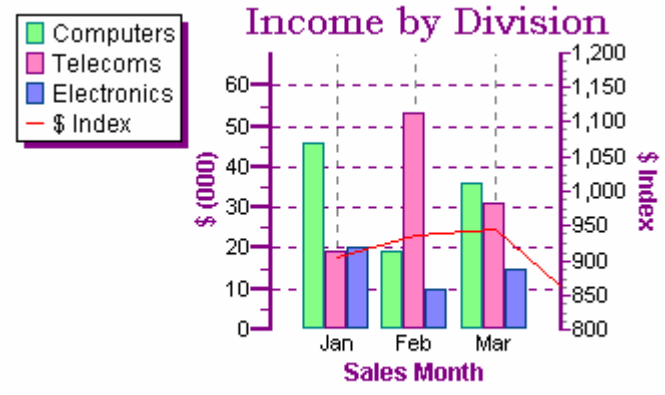


Example Series combination

Combining Seriestypes in one chart can add a great deal of information value. The following example shows the incomes by Division and puts the $ index in the same chart to measure the effect of that external influence on incomes (perhaps no influence at all in this example !!).

Fig. 8. -13.

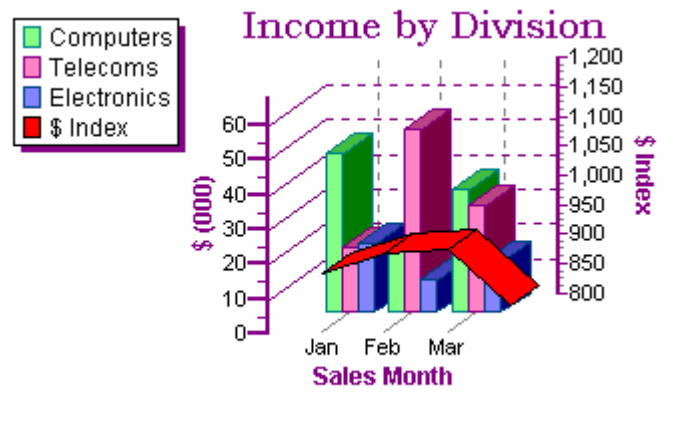
2D Combining Series types



You may combine Series in 3D Charts. The previous example is represented in 3D below. The chart looks attractive although a high degree of 3D perspective makes it more difficult to visualise the monthly $ index with Division incomes.

Fig. 8. -14.

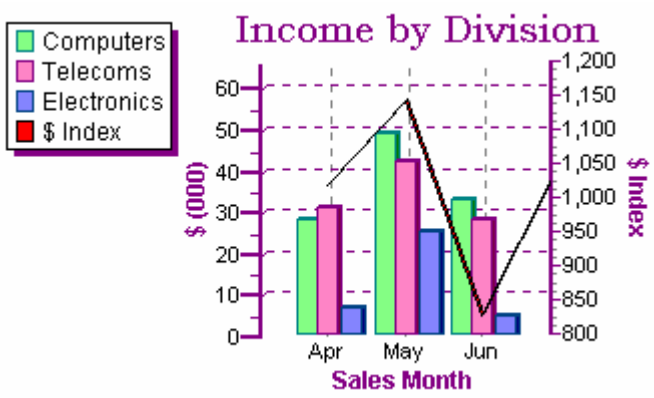
3D combining Series types (60% 3D)



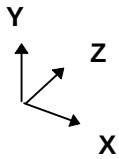
You could minimise the effect by reducing the extent of 'side' view (percentage 3D).

Fig. 8. -15.

3D combining Series types (6% 3D)



Or you could modify code so that at runtime the chart will put all Series in the same Z-plane. 3D charts have a property called Z-Order (see figure) which controls the depth position of each Series.



It is possible to put all Series on the same plane although we advise caution as the effect may be confusing depending on which Series types you are combining.

The following line of code (name of chart here **TChart1**) will put all Series in the same Z plane:

```
TChart1.Aspect.ApplyZOrder=False
```

9. Extended Series

There are 5 Extended Series Types included with TeeChart Pro Activex Control.

Candle

Fig. 9. -1.

2D Candle with moving average and volume

The Candle Series' properties include HighValues, LowValues, OpenValues and CloseValues and DateValues. The candle is ideally suited to tracking financial market information.

Example

If you look at the figure of the zoomed candle you can see how the financial information is tracked. White bars reflect the market rising, high end of the white bar being the day close. The red bars identify a fall in the market. The thinner lines show the day's high and lows.

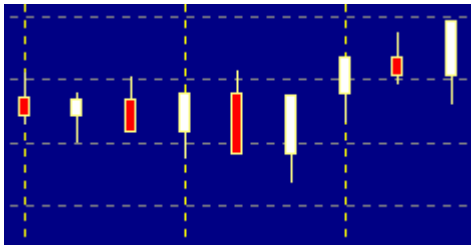


Fig. 9. -2.

Candle zoomed

The following code shows use of the AddOHLC method. This code fills the Series, evenly divided across the screen, with random data. You could substitute the random clauses with sources of you own data or connect your CandleSeries directly to a dataset via the Chart editor.

```
Dim tmpopen, tmp As Integer
With TChart1.Series(0)
  .Clear
  tmpopen = 1000 + Rnd(100)
  For t = 1 To 30
    tmp = Int(100 * Rnd - 50)
    .asCandle.AddCandle Date + t, tmpopen, tmpopen + 20, tmpopen - 20,
tmpopen + tmp
```

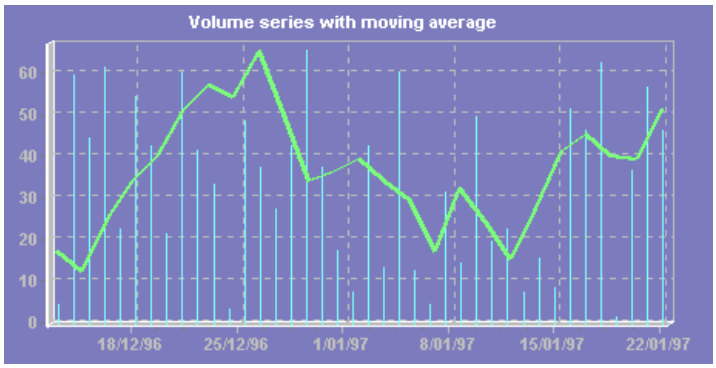
```
    tmpopen = tmpopen + tmp
  Next t
End With
```

Volume

The volume Series is another Series with origins in financial markets. It behaves very much like a bar Series except that each bar is represented as a thin line and the bars cannot be stacked.

Fig. 9. -3.

Volume Series with moving average



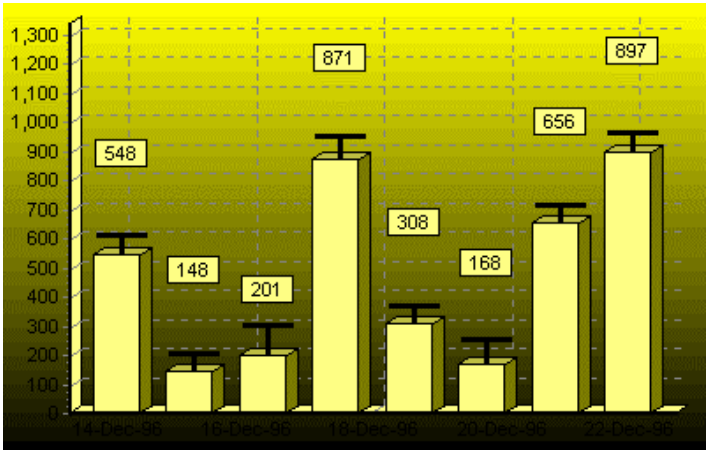
Errorbar

The height of the T on top of the bars of the Error Series show the size of the error.

Error bar Series may be applied to any data that has a real and estimated value, a success and failure level, etc.

Fig. 9. -4.

3D Error bar

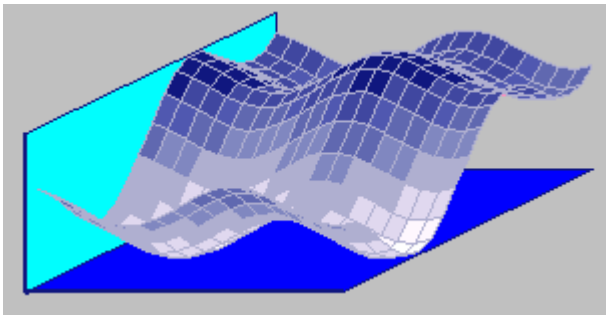


Surface

Surface Series use co-ordinates in 3 planes. TeeChart surface Series support null values as 'none' data points which appear as holes in the surface.

Fig. 9. -5.

3D Surface chart

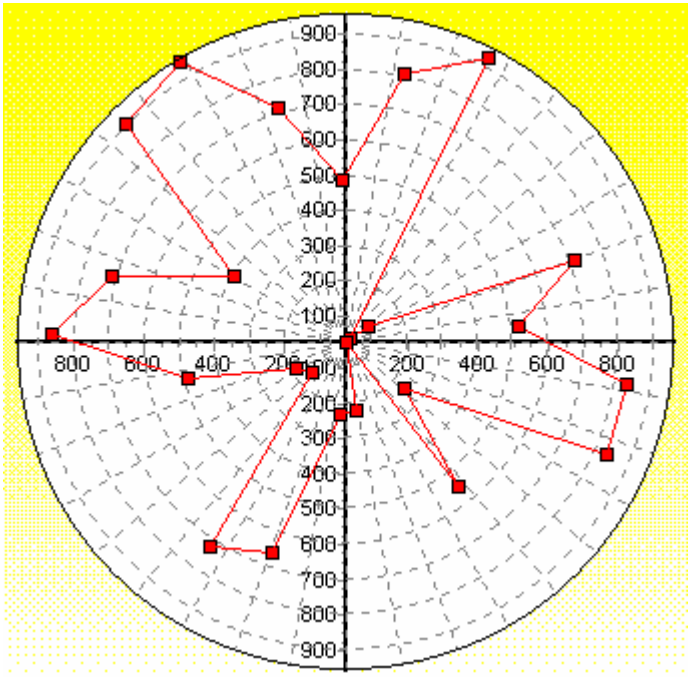


Polar

The polar Series plots XValues as angular rotation from 0º. The second variable, YValues are plotted as distance from the origin.

Fig. 9. -6.

2D Polar Series



10. Functions

Functions are independent components that use Series. When you add a new function, really what you are doing is adding a new Series (as any other) then applying a function definition. Normally that function works upon one or more other Series to give a dynamic result. For example if you define a line Series 'A' which has various values across one year and then you define a Series 'B' which contains the function Max(A) then you will be using Series 'A' as the input for the new Series. The Max(A) function (datasource of Series 'B') will scan the data from Series A to find the maximum value then plot its own data as that value. The Series type of the function is configurable, you may use any Series type to represent your function data provided that the Series type mix on the chart complies with the general rules for combining Series types - You may define a Function to use input Series from another chart as its Datasource.

Note
When you add a function from the gallery of the Chart Editor, by default you will add a LineSeries. You may change the type of Series afterwards to any other available Series type using the **Change** button.

In general, all functions work to the same basic rules. The properties of the underlying Series type (Line, Bar, etc.) apply to the function Series. Thus a function added as a Bar Series will have standard bar properties such as barwidth. There is one important property that distinguishes function Series from non-function Series, **Period**. Period applies to repetition of the function. For example, 'I want the average of all points of Series1' will present one flat line across the chart which is the average. 'I want the average of all points of Series1 by month' will draw an average for each month and will display as 12 different values across a chart of 1 year.

Important

Study the implications of period carefully before applying it to your chart Series. When you have **only 1 Series** in your function the function will, by **default, not apply period** and you must add it manually by code if you require period breakdown of the Series data. When you have **more than 1 Series** in your function the **default applied is period = 1**. If you wish to obtain any other period for the function you must add it manually by code. Period is based on number of points, we don't recommend the implementation of functions composed of Series with distinct point frequencies.

Adding a Function with the Chart Editor

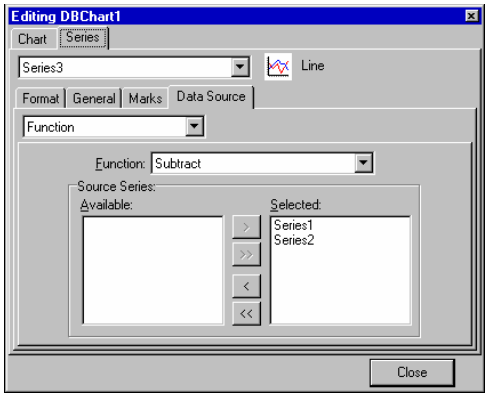
Functions may be added using the Chart editor at design time. Similarly to adding a Series you may use the TeeChart Gallery to choose the function. In the Gallery all functions are initially displayed as LineSeries functions - you may change Series type after initially adding the function.

When you have added the function go to the DataSource tab of the Series page. Here you may choose which input Series (or Series) to add to the function. The first Series in the selected listbox, reading from top to bottom, is the Series on which the operation is performed in the case of function which doesn't treat each member of the list homogeneously.

Eg. **Subtract**.

Fig. 10. -1.

Subtract function defined in the Chart editor



In the Chart editor screen you see Series3 being defined with inputs Series1 and Series2. The Series order in the list defines which Series is doing the subtraction.

Here: *Series3:= Series1 - Series2*

Step by step guide

To add a new Function with the Chart Editor follow these steps:

We will add the function Average to show the average value of an input Series.

1. Place a Chart on a form and add a new Series (The Section in this document "A TeeChart Quickstart Guide" will help you through the steps of adding a Series).

2. When you have a Series in your Chart, return to the Chart Editor and select the "Add" button on the first Chart page. The TeeChart Gallery will open. Select the "Functions" page.

3. Double click on the Function entitled "Average". This will add the Average function to your Chart and close the Gallery.

4. The Chart Editor will take you directly to the Datasource page for the new function. If you move elsewhere, return to the Series tab of the Chart Editor and select Series2 (the Function Series) from the dropdown combobox. Select the Datasource page for the Series.

5. The **Function** combobox will tell which FunctionType you have selected (this case "Average"). The **Source Series** framed area will show the first Series of your Chart in the left window. You may move it to the right window using the arrowbox, thus converting it to Datasource for the Function Series.

6. Close the Editor and put some code on your form to populate the first Series with data. Simplest would be, eg.:

```
TChart1.Series(0).FillSamplevalues 20
```

   Run the Project.

7. You will see the Average function drwaing a line across your Chart to display the average value of Series1.

8. You could add a button or scrollbar that modifies the period for the Function. Eg:

```
TChart1.Series(1).FunctionType.Period = 2
```

   That will alter the Average function to show the average for every 2 points of Series1.

Deleting a function with the Chart editor

You may delete the Series you have added as carrier for the function (delete Series from the first page of the Chart editor) or you may redefine the Series as having a different datasource in the DataSource tab of the Series page.

Changing a FunctionType with the Chart editor

The option to change the FunctionType is in the DataSource tab of the Series page for the function Series. The drop down combobox contains a list of all function types - You may choose any from the list. Copy will directly copy the input Series (duplicating the Series).

Adding a Function by Code

1. Add a data Series to your Chart and populate it with data. You could achieve that by using the Chart Editor at design time or by putting code behind a Command button, eg.:

```
With TChart1
  .AddSeries(scBar)
  .Series(0).FillSamplevalues 20
End with
```

2. Add another Command button to the project and add the following code:

```
With TChart1
```

```
    .AddSeries(scLine)  'new Series to form basis for the function
    .Series(1).SetFunction(tfAverage) 'apply the function to the Series
    .Series(1).Datasource="Series0"  'add the first Series as Datasource
  End with
```

3. Run the project and click the button to add the first Series, then the 2nd button to add the Function. you will see an average of the Bar Series display on the Chart.

4. You may change the average period by code, eg.:

```
TChart1.Series(1).FunctionType.Period = 2
```

That will alter the Average function to show the average for every 2 points of Series1. See the section entitled 'Period' for more information about this property.

Deleting a function by code

To delete a Function Series, delete the Series.

```
TChart1.RemoveSeries(1)
```

Period

You may find the Period property extremely useful when working with functions. It is used to define the frequency for which the function re-calculates.

Example

We have 6 data points (eg. bars of a Bar Series) with values:

3, 8, 6, 2, 9 and 12

We define a function Series with Period 0 (default when only one Series is input to the Function) the average drawn is:

6.667

With Period set to 2 we get 3 values of average as output from the function:

5.5, 4 and 10.5

These values will plot centrally in their period range, ie. The 1st value between bars 1 and 2 of the input Series, 2nd value between bars 3 and 4, etc..

You may define Period by selecting the function in the Chart Editor or you may modify Period at runtime using FunctionType.

Eg. Where Series(1) is the function Series:

```
TChart1.Series(1).FunctionType.Period=2
```

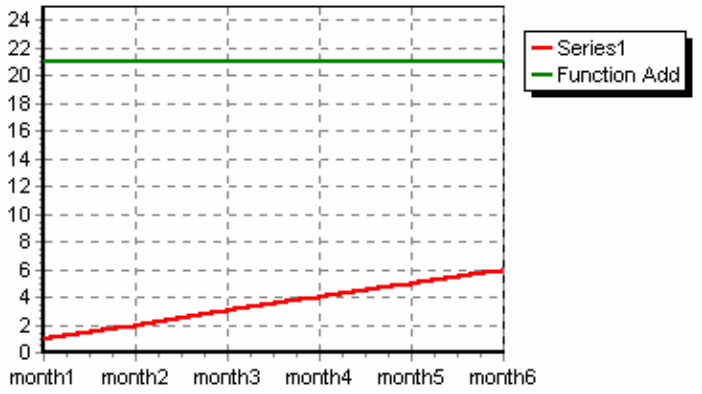
Standard functions

Add

The Add function adds data from one or more Series. If we create a line Series 'Series1', create a line Series 'Function Add' and define Series 'Function Add' as Add of Series1 and do nothing more we will obtain a chart with Series1 displayed and 'Function Add' as one flat line which is the sum of all values of Series1. In the figure the total of 1 + 2 + 3 + 4 + 5 + 6 = 21.

Fig. 10. -1.

2D Add function with 1 Series input



We can modify the Series 'Function Add' to represent Add of Series1 by 2 point groupings (1+2), (3+4), (5+6). We use the period property (Chart Editor in the Data source page). Coded it looks like this:

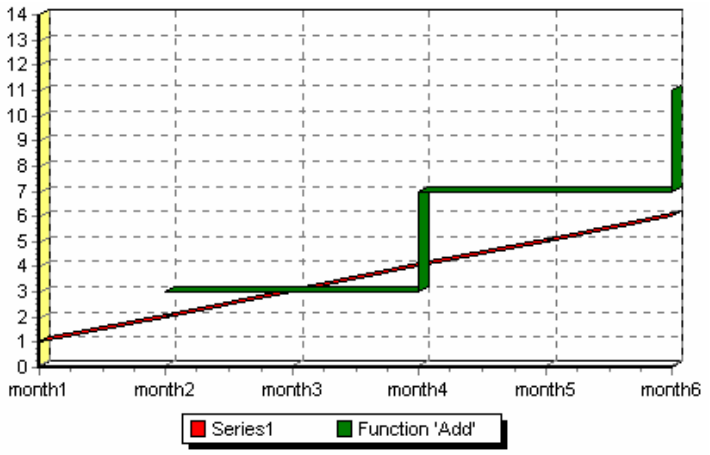
```
TChart1.Series(2).FunctionType.Period=2

'{where Series(2) is the functionSeries}
```

Defining the period as 2, sets the Add function to add every 2 points. The period property adds enormous value to function Series.

Fig. 10. -2.

Add function with period = 2



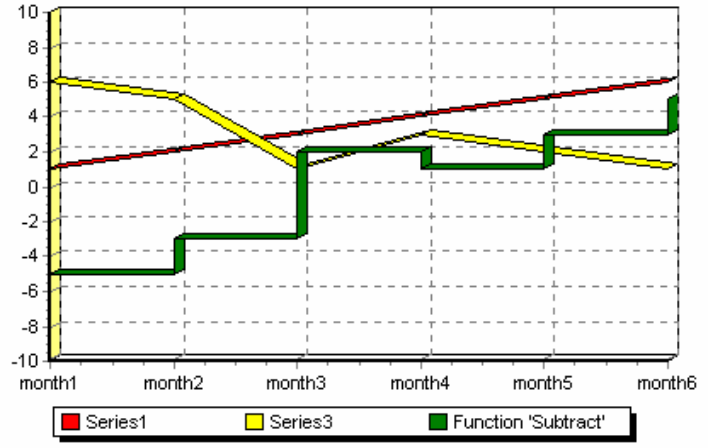
Subtract

Subtract requires 2 input Series. With more than one Series in the function the default period sets to 1 axis point. the 2nd Series will be subtracted from the 1st Series in the list.

The following chart is defined with ApplyZOrder:=False which forces all Series to draw in the same 3-Dimensional plane - The resulting Series overlay (a sort of optical illusion) in the chart depends on the Series paint order.

Fig. 10. -3.

Subtract function

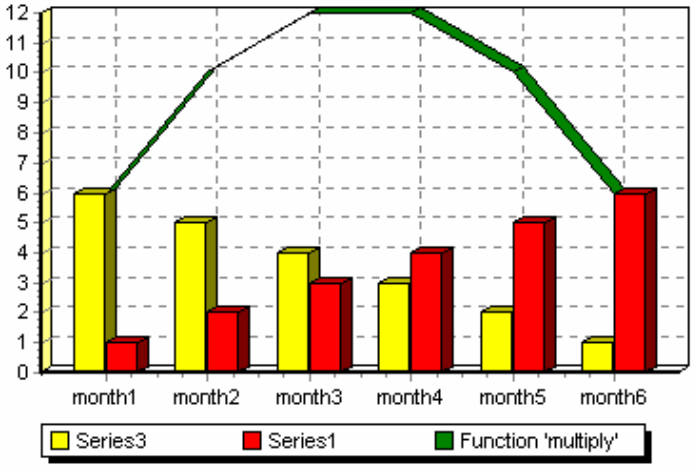


Multiply

The default period for the function 'multiply' is 1. You may add as many Series as you wish to the multiply function.

Fig. 10. -4.

Multiply function



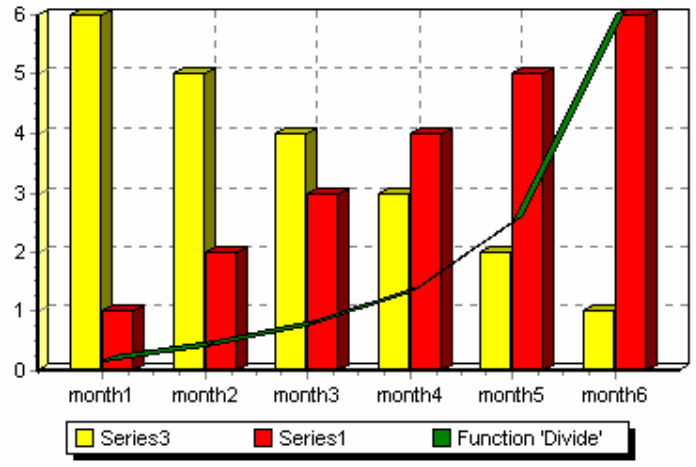
Divide

As divide requires at least 2 input Series the default period for the divide function is 1. The 2nd Series in the list is the denominator.

If you add more than 2 Series then the 1st will be divided by the 2nd then that is divided by the third, etc.…

Fig. 10. -5.

Divide function

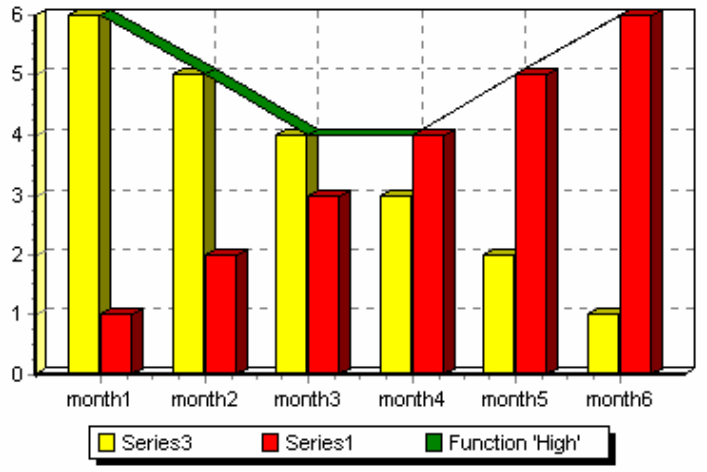


High

High accepts multiple input Series and will always display the highest point between those Series at each period point. (1 Series default period 0, 2 or more Series default period 1).

Fig. 10. -6.

High function

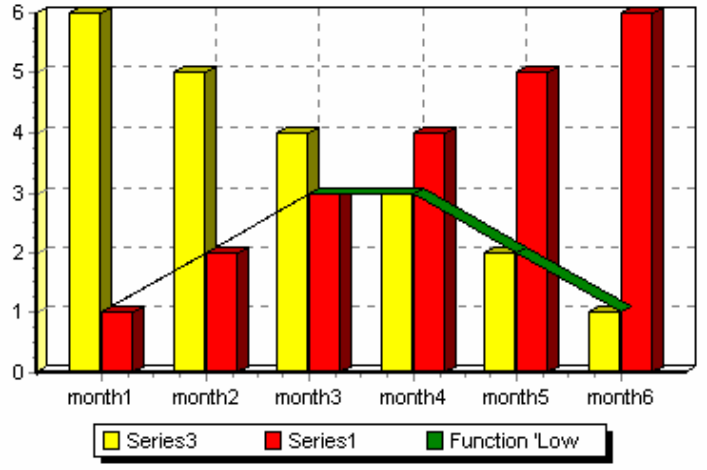


Low

Low accepts multiple input Series. It will always display the lowest point between those Series at each period point. (1 Series default period 0, 2 or more Series default period 1).

Fig. 10. -7.

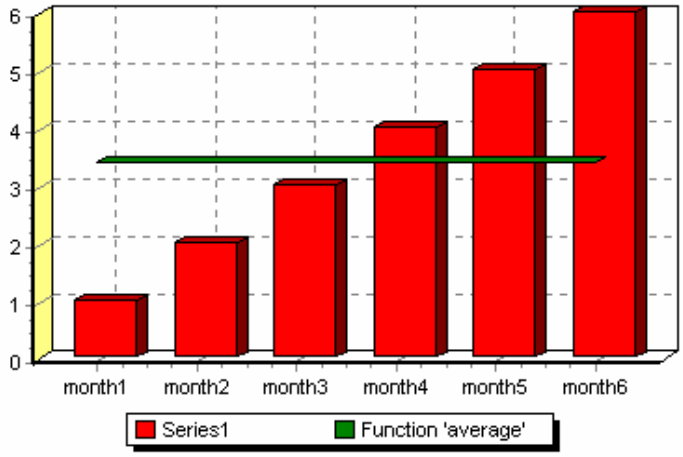
Low function



Average

Fig. 10. -8.

Average function



The default period for average with one Series is 0 (all) which will give you the average for that Series across the whole chart. If you have more than one Series the period will be 1 axis point.

You may change the period to alter the frequency of the average curve.

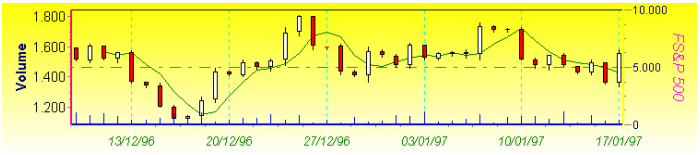
Extended Functions

Moving average

The moving average function permits you to track the current average as your data charts. You may define the period over which the moving average steps.

Fig. 10. -9.

Moving average applied to track data in a candle Series



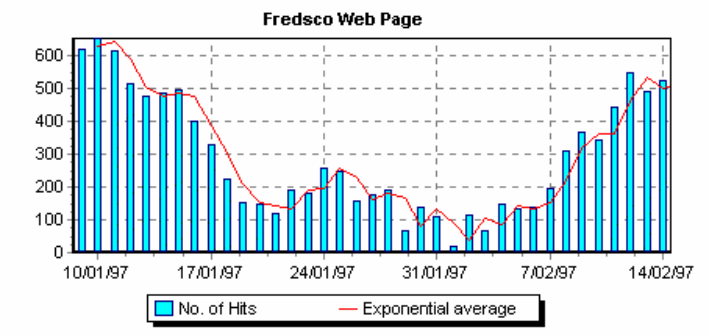
Exponential Average

The exponential average is similar to a moving average. It has a weight factor to add importance to more recent data.

The diagram shows an exponential average with weighting 0.2.

Fig. 10. -2.

2D

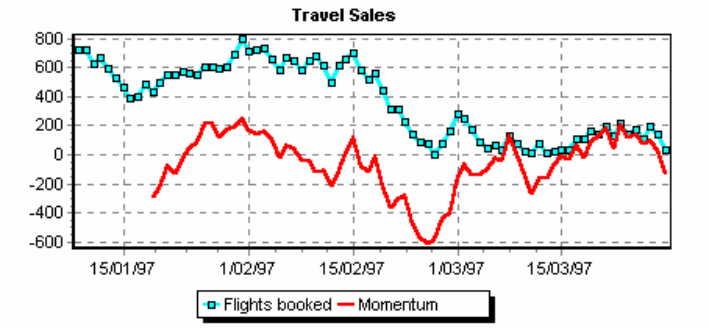


Momentum

A momentum Series is defined usng period. The curve takes the last value of the period and subtracts the first value.

Fig. 10. -3.

2D Momentum Series Period 10.

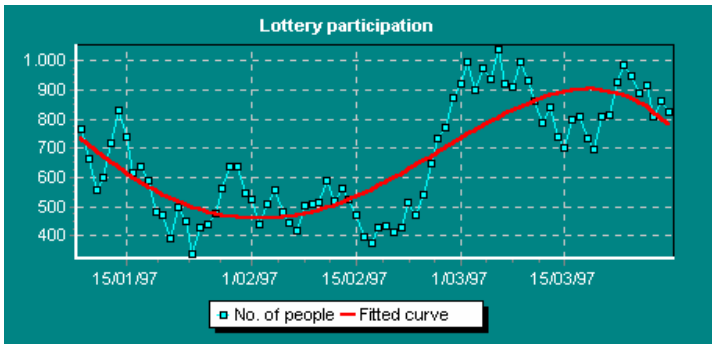


Fitted Curve

The CurveFitting Function performs a polynomical gaussian calculation on the underlying Series data to draw a smooth curve over the original points:

Fig. 10. -5.

2D Curve fit



R.S.I.

Relative Strength Index (RSI) is often used in financial applications

Fig. 10. -4.

2D RSI curve calculating over the last 20 data points of a candle series

11. Working with charts and series

You have seen how to create a chart, add Series and populate them. Now we'll take an introductory look at how to access the chart data at runtime. This chapter highlights some of the more common ways to program access to chart data. You will find more detailed information and example code on these and other advanced subjects in the TeeChart tutorials.

Click events

Chart OnClickSeries

The TChart event, OnClickSeries, permits access to any active Series in your chart. Put the following code into your project. To access the event and add the procedure definition to your project in Visual Basic, double-click on your chart. The Left Object Combobox will show the name of the TChart and the right will list the event procedures associated with the Chart. Select the event OnClickSeries.

```
Private Sub TChart1_OnClickSeries(ByVal Series As TeeChart.ISeries, ByVal
ValueIndex As Long, ByVal Button As TeeChart.EMouseButton, ByVal Shift As
TeeChart.EShiftState, ByVal X As Long, ByVal Y As Long)
    MsgBox (" Clicked Series: " + Series.Name + " at point: " +
Str$(ValueIndex))
End Sub
```

The valueindex refers to the index of the Series data point (point, bar, etc.) within the chart. You may use it to access the X and Y value. For example:

```
MsgBox ("Clicked Series: " + Series.Name + " at point: " + _
Str$(Series.XValues.Value(ValueIndex)) + "," + _
Str$(Series.YValues.Value(ValueIndex)))
```

Chart OnClick

You may use the OnClick event of the Chart to get the same information.

```
Private Sub TChart1_OnClick()
Dim t As Integer
  For t = 0 To TChart1.SeriesCount - 1
    If TChart1.Series(t).GetMousePoint <> -1 Then
      MsgBox (" Clicked Series: " + TChart1.Series(t).Name + " at _
 point: " + Str$(TChart1.Series(t).GetMousePoint))
    End If
  Next t
End Sub
```

Custom drawing on the chart

If you require to add textboxes or other shapes relative to chart axis the Shape Series may be the best choice. If the shape Series doesn't match your specific requirement you may draw your own lines and/or shapes on the chart.

Calculating Co-ordinates

This chapter explains how to convert from Point co-ordinates to pixels and vice-versa. Also how to determine the exact co-ordinates for each graphics element in Chart components.

Point Values are expressed in user custom scales. Chart Axis are responsible for converting point values into suitable X and Y screen pixel co-ordinates for displaying them.

Both Axis and Series components have several functions to convert from screen co-ordinates (in pixels) to Axis or points values (in user-defined units).

The difference between using Axis or Series conversion functions is that Axis will only interpret co-ordinates for the topmost position in 3D mode, while Series will adjust co-ordinates to their Z order position.

Note: Using conversion functions is only valid after a Chart component has been drawn, either to screen or to a private hidden Canvas.

Axis Methods

You can calculate the screen position for a given value using any Axis:

```
Dim MyPos As Integer
MyPos = TChart1.Axis.Left.CalcYPosValue(100)
MsgBox (MyPos)
```

MyPos holds now the pixel co-ordinate for "100.0" on TChart1 Left Axis (Vertical), being "100.0" a floating value in Axis scales.

You can use this co-ordinate to custom draw or to calculate mouse clicks. See the Tutorial "Custom Drawing".

If you want to convert from pixel co-ordinates to Axis values, you can use the opposite function:

```
Dim MyPos As Integer
MyPos = TChart1.Axis.Left.CalcPosPoint(100)
MsgBox (MyPos
```

MyValue holds now the Axis value co-ordinate for “100” on Chart1. Left Axis (Vertical), being “100” a screen pixel co-ordinate.

Note: Pixel co-ordinates start from 0, being 0,0 the Chart origin point. This is valid for screen, but when drawing to metafiles, drawing to custom canvases or printing, Chart origin point can optionally be different to 0,0.

You could use CalcYPos and CalcXPos. When drawing using the XPos and YPos co-ordinates remember that the co-ordinate 0,0 is Top,Left of the Chart rectangle.

The following example draws a line from an arbitary point from the Y-axis across the chart. Note the use of canvas properties.

```
Private Sub Command3_Click()
Dim MyHalfwayPoint, YPosition As Integer
With TChart1.Series(0).YValues
   MyHalfwayPoint = CInt(((.Maximum - .Minimum) * 0.5) + .Minimum)
   ' then calculate the Screen Pixel co-ordinate of the above value
   YPosition = TChart1.Axis.Left.CalcYPosValue(MyHalfwayPoint)
   With TChart1.Canvas
     ' change pen and draw the line avoiding the
     ' 3D areas and axis of the chart - Height3D,Width3D
     .Pen.Width = 3
     .Pen.Style = psSolid
     .Pen.Color = vbBlack
     With TChart1
      .Canvas.MoveTo .Axis.Bottom.CalcXPosValue(0), YPosition
      .Canvas.LineTo .Axis.Bottom.CalcXPosValue(0) + .Aspect.Width3D, _
              YPosition - .Aspect.Height3D
      .Canvas.LineTo .Axis.Bottom.CalcXPosValue(.Axis.Bottom.Maximum)+ _
             .Aspect.Width3D, YPosition - .Aspect.Height3D
     End With
  End With
End With
End Sub
```

Series Methods

Series have similar methods for converting co-ordinates to point values and vice-versa. The main difference is that by using the Series method you don’t need to know the exact Axis component for calculations.
This is a big advantage when having Series associated to Right or Top Axis, or multiple Series associated to each Axis.

This code calculates where in the screen the Series(0) point with value 1000 is located:

```
    Dim MyPos as Integer
    MyPos = TChart1.Series(0).CalcXPosValue( 1000 )
```

or...

```
    MyXPos =
 TChart1.Series(0).CalcXPosValue(TChart1.Series(0).XValues.Value(0))
  ' <-- first point }
```

Chart Canvas

Chart.Canvas is derived from a standard Delphi Canvas. You may control the appearance of your chart using Canvas properties.

The following example divides the area of the backdrop of the Chart rectangle into 5 equal segments and colours them according to the colour array. Put the code in the TChart OnBeforeDrawSeries event to see the result.

```
Dim MyColors(5) As ColorConstants
Dim t, partial, tmpLeft, tmpTop, tmpRight, tmpBottom As Integer
    MyColors(0) = RGB(215, 0, 0)
    MyColors(1) = RGB(255, 128, 0)
    MyColors(2) = RGB(128, 128, 255)
    MyColors(3) = RGB(0, 0, 153)
    MyColors(4) = RGB(215, 0, 0)

 With TChart1
   ' setup the draw area
   tmpTop = TChart1.Axis.Left.CalcYPosValue(TChart1.Axis.Left.Maximum)
   tmpBottom = TChart1.Axis.Left.CalcYPosValue _
               (TChart1.Axis.Left.Minimum)
   tmpLeft = TChart1.Axis.Bottom.CalcXPosValue _
             (TChart1.Axis.Bottom.Minimum)
   tmpRight = tmpLeft
   ' we will divide the total chart width by 5
   partial = (TChart1.Axis.Bottom.CalcXPosValue _
              (TChart1.Axis.Bottom.Maximum) - tmpLeft) / 5

   ' change the brush style
   .Canvas.Brush.Style = bsSolid
   .Canvas.Pen.Style = psSolid

   ' for each section,fill with a specific color
   For t = 0 To 4
     ' adjust the rectangle dimension
     tmpRight = tmpRight + partial
     ' set the brush color
     .Canvas.Brush.Color = MyColors(t)
     .Canvas.Pen.Color = MyColors(t)
     ' paint !!!
     .Canvas.Rectangle (tmpLeft + TChart1.Aspect.Width3D), _
                       (tmpTop - TChart1.Aspect.Height3D), _
                       (tmpRight + TChart1.Aspect.Width3D), _
                       (tmpBottom - TChart1.Aspect.Height3D)
     ' adjust rectangle
     tmpLeft = tmpRight
   Next t
 End With
```

Repainting

You should call Chart1.Repaint or Series1.Repaint to force a Chart component to draw again.

When to draw ?

Best place to custom draw over a Chart component is at

TChart1.OnAfterDraw event, which is fired each time the Chart component is redrawn, just before copying the internal bitmap to screen.

The equivalent of drawing before the Chart is drawn is to use the OnBeforeDrawSeries event, after Chart walls and background have been painted.

Setting Axis scales

Chart components have four axes: Left, Right, Top and Bottom.

Axes are responsible for calculating pixel co-ordinates for Series points and to allow any valid range so scroll and zoom can be always performed. As new Series are inserted, or new points are added to the Series, Axis recalculate, by default, their Minimum and Maximum values.

You can turn off automatic recalculation of Axis scales by setting the Automatic property to false:

```
TChart1.Axis.Left.Automatic = False
```

Also, both the Axis Minimum and Axis Maximum values can optionally be automatic or not (independently of each other).

```
TChart1.Axis.Left.AutomaticMaximum = False
TChart1.Axis.Left.AutomaticMinimum = True
```

You can change Axis scales using the Minimum and Maximum properties:

```
With TChart1.Axis.Left
    Automatic = False
    Minimum = 0
    Maximum = 10000
End with
```

or using the Axis SetMinMax method:

```
TChart1.Axis.Left.SetMinMax( 0, 10000 )
```

DateTime Axis

Note:
An Axis contains DateTime scales when the associated Series components have XValues.DateTime or YValues.DateTime properties True. There is no DateTime property for Axis.

Changing scales in DateTime Axis is the same as for non-datetime values:

```
With TChart1.LeftAxis
   begin
       Automatic = False
       Minimum = DateValue("1,2,1998")
       Maximum = DateValue("28,2,1998")
   End with
```

Logarithmic Axis

An Axis can be to Logarithmic only if Axis Minimum and Maximum are greater than or equal to zero. This is the only difference between setting linear and logarithmic scales.

Note:
Axis Labels are not displayed in logarithmic increments. You can generate custom Axis Labels (see chapter below).

Inverted Axis

An Axis can be Inverted so Axis Minimum and Maximum are swapped. We suggest you use Inverted:=True as little as possible as it can give misleading results due to its (normally) rarity of use.

Axis Styles & Increment

Axis can be displayed in several ways, with tick lines or without,
with grids or without, with Labels or without, and you can customise all formatting properties such as colours, fonts and pen styles. The Axis Increment property controls the number of grid lines and labels and the distance between them.

By default it's zero, meaning Axis will automatically calculate the Labels increment. Setting the Axis Increment property is independent of setting Axis scales. You can have automatic Axis Maximum and Minimum and a fixed Increment.

The Increment property grows automatically until all Axis Labels can be displayed. If you don't want this automatic feature, set the Axis LabelsSeparation to zero:

```
TChart1.Axis.Left.LabelsSeparation =  0
```

Warning:
When LabelsSeparation is zero, no checking is performed against
labels size, so you must take care labels will not overlap each other.

The following code sets the vertical Axis Increment to 30:

```
TChart1.Axis.Left.Increment= 30
```

By default, the first Axis label starts at nearest Increment. Setting RoundFirstLabel to False makes labels to start at Axis Maximum:

```
TChart1.Axis.Left.RoundFirstLabel = False
```

DateTime Increment

Use the predefined DateTimeStep array of constants to determine the Axis Increment on DateTime axis:

```
TChart1.Axis.Bottom.Increment = TChart1.GetDateTimeStep( dtOneMonth )
```

And set the ExactDateTime property to True if you want Axis Labels to be at exact datetime boundaries, for example at 1st day of month.

```
TChart1.Axis.Bottom.ExactDateTime = True
```

Note:
Logarithmic axis use the Increment property as linear.

Grid lines

Axis Grid lines are displayed at each Increment position, or at each Axis Label position. The Axis TickOnLabelsOnly property controls this feature:

```
TChart1.Axis.Bottom.TickOnLabelsOnly = False
```

Axis Labels

There are several Axis labelling styles. The Axis LabelStyle property control axis labelling modes:

```
TChart1.Axis.Bottom.Labels.Style = talValue
```

Possible values are:

Labels display Axis scales
Labels display Series Point Marks
Labels display Series XLabels
No labels displayed
Style is automatically calculated

When LabelStyle is talText, TeeChart will set it automatically to talValue if Series have no XLabels.

For talMark and talText styles, axis labels are displayed exactly at Series point positions, thus not using the axis Increment property.

For advanced techniques for working with Axis labels, see the TeeChart Pro tutorials.

CustomDraw (Axis)

CustomDraw adds more axis to your chart - As many as you wish. CustomDraw adds an axis as a 'copy' of an existing axis at a new location.

The following example populates a line Series with random data. And creates 2 additional axis. There are 3 configurable locations PosLabels, PosTitle and PosAxis to place, respectively, the labels, title and the axis. The last parameter, GridVisible , which is boolean, defines whether to extend the axis grid to the new axis.

```
 Private Sub Command1_Click()
 Dim t As Integer
   For t = 0 To 20
     TChart1.Series(0).AddXY t, ((100 * Rnd) + 1) - ((Rnd * 70) + 1), _
                        "", vbRed
   Next t
 End Sub

Private Sub TChart1_OnBeforeDrawSeries()
Dim posaxis As Integer
  With TChart1
   ' When scrolling or on zoom always keep the gridlines enclosed in the
   ' Chart rectangle
   .Canvas.ClipRectangle .Axis.Left.Position, _
   (.Axis.Left.CalcYPosValue(.Axis.Left.Maximum)), _
   (.Axis.Bottom.CalcXPosValue(.Axis.Bottom.Maximum)), _
   .Axis.Bottom.Position
   ' Always draw the 2nd vertical Axis at the middle point of the Chart
   posaxis = (.Axis.Left.Position) _
             + (((.Axis.Bottom.CalcXPosValue(.Axis.Bottom.Maximum)) _
             - (.Axis.Left.Position)) * 0.5)
   .Axis.Left.CustomDraw posaxis - 10, posaxis - 20, posaxis, True
   ' Draw the 2nd Horizontal axis at the level of "10" on the vertical
   ' axis
   posaxis = (.Axis.Left.CalcYPosValue(10))
   .Axis.Bottom.CustomDraw posaxis + 10, posaxis + 40, posaxis, True
   .Canvas.UnClipRectangle 'Do not carry over cliprectangle
  End With
End sub
```

Series manipulation

This chapter documents how to manipulate Series and how to manipulate Series points and other Series internal data.

Adding Series at runtime.

You may add a new Series to a Chart using the following code:

```
 TChart1.AddSeries(scLine)
```

ESeriesClass scLine will add a LineSeries. Alternatively add a BarSeries (scBar), AreaSeries (scArea), etc. See the online help for the full options of ESeriesClass. {Class references:

Series array

Chart components store all Series in a Series list.

You have read-only access to this list by using the Series array property:

```
TChart1.Series( 0 ) ' is the first Series in the Chart.
```

SeriesCount property

The TChart1.SeriesCount property returns the number of Series in the Series list.

You can use SeriesCount to traverse all Chart Series:

```
for t = 0 to TChart1.SeriesCount - 1
  With TChart1.Series(t) do
   .Color = vbBlue
  End with
Next t
```

Deleting Series

Series can be hidden in 2 ways:

A) Setting the Series Active property:

```
TChart1.Series(0).Active = False
```

B) Destroying the Series completely:

```
TChart1.RemoveSeries(0)
```

Changing the Series Z order at runtime.

In 3D mode (when Chart1.View3D is True), all Series are assigned a Z order position. That is, the order where Series will be drawn, starting with the far most Series on the Chart 3D space.

You can control the order Series will be drawn, using this method:

```
TChart1.ExchangeSeries( 0, 1 )
```

Adding Points

Every Series type has, at least, 2 values for each point. These values are designed as X and Y point co-ordinates.

Note: Values are defined as "Double" floating point variables.

Extended Series types have more than 2 values, like BubbleSeries has X, Y and Radius values for each point.

So, each Series type has its appropriate method to add points, although the most common Series types like Line, Bar, Point and Area share the generic AddY method to add points.

The following code empties a Pie Series and adds some sample values to it:

```
TChart1.Series(0).Clear
TChart1.Series(0).Add   1234, "USA", vbBlue
TChart1.Series(0).Add   2001, "Europe", vbRed
```

For extended Series types, please refer to each specific Series reference to know which method should be used to add points.

Null Values

In some circumstances you might have no values for specific points. You should then add these points as "zero" or add them as "null" values.

Null values will not be displayed, while "zero" values will be displayed as usual.

The following code adds several points and a null point:

```
With TChart1.Series(0)
  .Clear
  .Add 10, "John", vbGreen
  .Add 20, "Anne", vbYellow
  .Add 15, "Thomas", vbRed
  .AddNull "Peter"
  .Add 25, "Tony", vbBlue
  .Add 20, "Mike", vbCyan
End with
```

Each Series type will handle null values in different ways.
Bar, HorizBar, Line, Area and Point do not display null points.
PieSeries use null values as “zero”.

Controlling Points Order

Points can be optionally sorted either by their X values or Y values. The Series.XValues and Series.YValues Order property controls the points Order:

```
TChart1.Series(0).XValues.Order = loAscending
```

Possible values are: loNone, loAscending or loDescending.

By default, XValues Order is set to loAscending, and YValues Order to loNone, which means new added points are ordered by their X co-ordinate. For non XY charts, the X co-ordinate is always the point position, starting from zero.

The point Order is used by Series components to draw their points.

Note: Order must be set before points are added to the Series.
You can change the Order property at run-time, followed by a call to Sort method:

Example:

Drop a TChart onto a Form, add a Line Series.
Drop a TButton and assign this code to Button1Click:

```
With TChart1.Series(0)
  .Add 5, "", clTeeColor
  .Add 2, "", clTeeColor
  .Add 3, "", clTeeColor
  .Add 9, "", clTeeColor
End with
```

Now drop another TButton and add this code to Button2Click:

```
With TChart1.Series(0)
  .YValues.Order=loAscending
  .YValues.Sort
End with
TChart1.Repaint
```

Now execute and press Button1 to fill Series1, and press Button2 to see Series1 points drawn on Series1 YValues Ascending order, but having the original X co-ordinates.

Note: If you aren’t using X co-ordinates, one more line of code is required.

Drop a new TButton (Button3) and add this code to Button3Click:

```
TChart1.Series(0).XValues.FillSequence
TChart1.Repaint
```

Now points will be “re-numbered”, starting from zero among Series1 XValues axis. This will re-order points so now they will be drawn as if they were originally added to the Series in their new order.

This “two-step” point sorting allows Line Series to draw with vertical orientation.

XY Points

Adding X co-ordinates to points makes Series components calculate user specific horizontal point positions.

Note: Bar charts can be difficult to interpret with X co-ordinates.
Pie Series do not allow X co-ordinates.

To add X co-ordinates, simply use the AddXY method:

Drop a TChart on a Form, add a Point Series:

```
With TChart1.Series(0)
  .Clear
  .AddXY  10, 10, "Barcelona", clBlue
  .AddXY  1, 10, "San Francisco", clRed
End with
```

Note: If you use a Bubble Series, use the BubbleSeries AddBubble method.

Remember to set XValues.Order to loNone if don’t want points to be sorted on their X co-ordinate.

Point Limits

Maximum 134217727 points per Series and same number of Series per Chart.

Deleting Points

Simply call the Series.Delete method, passing the point index as the argument. Point index starts at zero.

```
TChart1.Series(0).Delete( 0 )
' removes the first point in TChart1.Series(0)

TChart1.Series(0).Delete(TChart1.Series(0).Count - 1 )
' removes the last point in TChart1.Series(0)
```

An exception “List out of bounds” will be raised when attempting to Delete a non-existing point, so always check there are enough points in the Series prior to delete them:

```
if TChart1.Series(0).Count > MyIndex then
  TChart1.Series(0).Delete( MyIndex )
End if
```

Calling Delete forces re-calculation of Functions and repainting of the Chart.

Retrieving and modifying Points

Once points have been added, you can retrieve their co-ordinates or change them.

The XValues and YValues array properties can be used:

```
Dim MyValue as Double

MyValue = TChart1.Series(0).YValues(0).Value
' retrieves the first Y value }
```

You can traverse these arrays to perform calculations:

```
Dim MyTotal as Double
Dim t as Integer

MyTotal = 0
for t= 0 to TChart1.Series(0).Count - 1
  MyTotal = MyTotal + TChart1.Series(0).YValues.Value(t)
Next t

   Msgbox(Str$( MyTotal ) )
```

Extended Series types have additional array properties, such as BubbleSeries RadiusValues. You can access these properties in the same way as with XValues or YValues arrays:

```
if TChart1.Series(0).asBubble.RadiusValues.Value(Index) > 100 then .....
```

Modifying point values can be performed using the above properties:

```
TChart1.Series(0).YValues.Value(0) = _
TChart1.Series(0).YValues.Value(0) + 1
TChart1.Series(0).RefreshSeries
```

Locating Points

The XValues and YValues Locate function searches a specific value in the List and, if found, returns the value Index, starting from zero.

```
Dim MyIndex as Integer

MyIndex = TChart1.Series(0).YValues.Locate( 123 )

if MyIndex = -1 then
  Msgbox("  123 not found in Series1 !! ")
else
  Msgbox("  123 is the "+Str$( MyIndex+1 )+"th point in Series1 !!")
End if
```

Point Statistics

XValues and YValues properties maintain the following statistical figures:

Total	The sum of all values in the list.
TotalABS	The sum of all values as absolute (positive).
MaxValue	The maximum value in the list.
MinValue	The minimum value in the list.

You can call manually to RecalcMinMax method to recalculate MinValue and MaxValue. Total and TotalABS are maintained automatically.

These values are used to speed up several Axis scaling calculations and are used as helpers for percent calculations.

The ValueList Class has several other methods and properties to manipulate point values. Please refer to online help documentation.

Notifications

Whenever points are added, deleted or modified, the Series generates internal notification events. These events are used to recalculate Series that depend on other Series points.

```
TChart1.Series(0).RefreshSeries
```

Will manually force dependant Series to recalculate their points.

Point Colours

All Series maintain an internal List of colours. One for each point in the Series.

You can access this list with PointColor array property, to retrieve or change point colours:

```
Var MyColor : TColor ;

MyColor = TChart1.Series(0).PointColor(0)

TChart1.Series(0).PointColor(1) = vbBlue
```

TeeChart defines a generic colour constant named: clTeeColor.
Points with clTeeColor colour will be drawn using the SeriesColor colour.

Visual Basic predefines constants for basic colours (vbBlue, vbRed, etc.). Colours can also be expressed in RGB format. Using a video colour depth of 16k colours or more results in better colour matching.

Point Labels

Each point has an associated text, called PointLabel, and declared as a string.

Point Labels are used in Axis Labels, Chart Legend and Point Marks.

Labels are stored at Series PointLabel array property.

You can access and modify PointLabel point texts:

```
TChart1.Series(0).PointLabel(0) = "Sales"
```

Changing the Series type at runtime - Advanced

Every chart type corresponds to a different Component Class.
Changing a Series type involves changing the Series Component Class.

That means a new Series of the new class must be created, old Series properties should be assigned to the new instance, and finally the old Series must be destroyed.

All this happens automatically when you manually change a Series type at design-time using the Chart Editor Dialog and the Gallery.

You can change a Series type at run-time calling:

eg.

```
TChart1.ChangeSeriesType 0, scBar
'Changes the existing 1st Series to a Bar Series
```

Warning:
If the new Series type does not share the same number of variables (ie a Bar Series has X and Y (2 variables), a CandleSeries has High,Open,Low,Close - (4 variables)) then the new Series type will not successfully paint.

Series type specific properties and methods

TeeChart Pro Activex Control uses class inheritance, internally, to provide a homogeneous environment for working with differing Series Types. This is a powerful feature of TeeChart that offers you the flexibility to mix many different types of Series in the same Chart and is the underlying reason why, with TeeChart, you first add the Chart, then are able to add the various Series you require to the Chart instead of choosing from a limited list of Chart types.

Some programming languages (eg. Visual Basic) don't support Class inheritance thus TeeChart provides homogeneity by grouping all common Series properties and methods in the Series class, and all Series type specific properties and methods in Series type interfaces accessible as Series properties.

Example

Common Series properties are accessible via the Series class

```
TChart1.Series(0).Color = vbBlue
```

Series specific properties are available via the Series type's interface. The Candle Series is accessible via the TChart.Series(Index) as are all other Series in the Chart.

```
TChart1.Series(0).asCandle.AddCandle DateValue("2,12,97"), _
                                     110,102,119,114
```

You may combine these:

```
With TChart1.Series( 0 )
  .Clear
  .Title = "My Candle Series"
  .asCandle.UpCloseColor   = vbGreen
  .asCandle.DownCloseColor = vbYellow
  .asCandle.AddCandle DateValue("2,12,97"),110,102,119,114
end With
```

Series specific properties / methods are accessible via the Series properties:

asArea, asArrow, asBar, asBubble, etc. See the TeeChart online help for a full listing.

Printing Charts

This chapter explains how Charts are printed and which properties and methods are used to control the printing process.

Margins

When printing, you can specify which margins on the paper page should be left blank. The public PrintMargins property stores the desired paper margins expressed as percents of total page dimensions.

```
PrintMargins:=Rect(15,15,15,15); { default 15 % printing margins }
```

The TChart.ChartPrintRect:TRect function returns, after applying the printing margins, the space where the Chart component will be drawn expressed as logical canvas units (pixels or dots).

With PrintMargins you can define any area of any size inside the page.

Note:
When changing paper orientation, margins are recalculated.

Detail

Charts are printed in metafile (WMF in 16bit, EMF in 32bit) format.

Metafiles are scaleable vector formats, so a “wysiwyg” effect can be achieved if sending to the printer how a Chart looks on screen. However, you might want to exploit the printer’s bigger detail capabilities versus screen displays.

The TChart.Printer.Detail integer property controls how a chart is scaled when sending it to the printer. By default Detail is zero. Setting it to a negative value in the percentage range from zero to -100 makes the Chart proportionally bigger so there’s more space for Axis Labels. Smaller fonts are used as they will be clearer on paper than on screen.

Setting it to a positive value makes font sizes bigger.

Print, PrintLandscape....

Several methods exist to print a Chart component:

```
TChart.Printer.Print
TChart.Printer.PrintPortrait
TChart.Printer.PrintLandscape
```

The above methods will do the same, print a Chart on a new page and **eject** (form feed) the page:

```
TChart.Printer.Orientation(AOrientation:TPrinterOrientation)
```

Will Change the Printed page orientation.

```
TChart.Printer.Print
```

Will print a TChart in the current printer’s paper orientation.

Windows and printers limitations

Metafiles are very good as they are small and fast and scaleable.
Some limitations occur when using metafiles, as described in Microsoft Knowledge Base (www.mskb.com).

TeeChart inherits those limitations:

Clipping
Clipping is stored in physical co-ordinates in metafiles. This means moving or scaling a clipped metafile will not scale or move the clipping region, giving undesirable results.

TeeChart does not draw partial points, so no clipping. Drawing on zoomed charts will probably show partial points outside the Chart axis.

Rounding circles
As metafiles can be scaled, circles will be converted to ellipses when scaling.

Rotating fonts
Rotated fonts can not be exactly aligned on non-proportionally scaled metafiles.

Dotted Pen styles and Pen Width
Non solid pens (dots, dashes), can be drawn as solid with scaled metafiles.

Chart Zoom and Scroll

Scrolling and Zooming a Chart is simply setting its Axis scales to the desired values. After zooming or scrolling a Chart, all Series will repaint their points in their new positions.

Note: Pie Series can't be scrolled or zoomed. You can control Pie dimensions using Chart margins or Pie custom radius properties.

Zoom

Charts can be zoomed programmatically or by user interaction with mouse dragging. The Chart Zoom.Enable property controls if users can apply zoom:

```
TChart1.Zoom.Enable = True
```

Users can zoom drawing a rectangle around the Chart area they want to see in detail.

Note:
Dragging should be done from top / left to bottom down (default: Left mouse button).
Dragging in the opposite direction resets axis scales (no zoom).

As soon as users release the mouse button, TeeChart repaints to show the zoomed area.

Animated Zoom

You can control if TeeChart will calculate zoom positions immediately or it will be calculating zoom in short "steps" until reaching the desired zoom window.
This makes an "animated" zoom effect, which helps to identify better the zoomed area.
This code activates animated zoom:

```
TChart1.Zoom.Animated = True
```

Set the Zoom.AnimatedSteps property to the desired number of intermediate zooms:

```
TChart1.Zoom.AnimatedSteps= 5
```

Zooming by code

You can zoom in or zoom out a Chart using any of these methods:

Zoom.ZoomRect adjusts axis scales to show the rectangular parameter area. The rectangle is expressed in screen pixel co-ordinates. Rectangle areas inside TChart1 rectangle (area bounded by axes) will zoom in, while area outside rectangle will zoom out.

```
TChart1.Zoom.ZoomRect 100, 100, 200, 200
```

Undoing Zoom

The Zoom.Undo method resets axis scales to their automatic Minimum and Maximum values:

```
TChart1.Zoom.Undo
```

This will display all Series points, undoing any previous zoom in or zoom out operation, either by code or using the mouse.

Note:
If you want axis scales to be at specific values after undoing zoom, you can use the TChart OnUndoZoom event, documented below.

Zoom Events

The OnZoom event is triggered whenever zoom is applied to a TChart, either manually or programmatically:

```
Private Sub TChart1_OnZoom()
  Button1.Visible = True
  'make visible the "no-zoom" button
end sub
```

The OnUndoZoom event is called when undoing zoom, by code or by mouse.

Scrolling

Scrolling is very similar to zoom. Axis scales are incremented or decremented and the whole chart component is repainted to show Series points at their new positions.

The Chart Scroll.Enable property controls if users can scroll Chart contents by dragging the mouse. Its possible values are:

pmNone	No scroll is allowed.
pmHorizontal, pmVertical	Allow scroll only on these directions.
pmBoth	Allow complete scroll over all directions.

Example:

```
TChart1.Scroll.Enable = pmNone
  ' no scroll is allowed
```

You can programmatically scroll a chart, using the Axis Scroll method:

```
Procedure Scroll(Offset: Double; CheckLimits: Boolean)
```

Example:

```
TChart1.Axis.Bottom.Scroll  1000, True
```

The above code increments Bottom Axis scales by 1000. This is the same as doing:

```
With TChart1.Axis.Bottom
  .SetMinMax .Minimum+1000, .Maximum+1000
End with
```

and setting AxisBottom Automatic property to False.

The Chart will repaint and the horizontal bottom axis will be "scrolled" to the left a quantity of "1000" in axis scales.

The "CheckLimits" boolean parameter instructs the axis to scroll ONLY if there are more Series points in the scrolling direction.

Scroll event

The Chart OnScroll event is fired every time users scroll manually the Chart.

```
Private Sub TChart1_OnScroll()
    Label1.Caption = "This chart has scrolled ! "
end sub
```

Real-Time Charting and Speed

Two big rules apply to speed performance in real-time charting:

1. Plot as few points as possible
2. Use the fastest possible hardware.

Together, these two rules really make a big speed difference when drawing charts many times continuously.

Some other suggestions are:

- Use 2D. Three dimensional charts are slower to paint than 2D.
- Make Charts small. Bigger charts need more pixels to be filled.
- Remove Chart Legend and Titles when possible.
- Use default fonts and font sizes.
- Use FastLineSeries for fastest way to plot many points.
- Use solid pens and brushes styles.
- Avoid using circular shapes or circular bar styles.
- Don't use background bitmaps or gradient fill effects.
- Set Chart BevelInner and BevelOuter properties to bvNone
- When possible, set Chart1.AxisVisible to False to remove axis.
- Set your video mode resolution and colour depth to the optimum values, according to your video card.
- A combination of 800x600 x 256 colours can be faster than 1024x768 x 32k colours, on average video cards.
- Use Windows 95 or Windows NT with accelerated drivers for your video card.

Index

Thank You and *Good TeeCharting* !!!

teeMach SL
P.O. Box 22262
08080 Barcelona
Tel. +34 7 259 71 61
Fax. + 34 3 423 59 82
email: info@teemach.com
www: www.teemach.com