

Definitive Guide to Excel VBA, Second Edition

MICHAEL KOFLER
TRANSLATED BY DAVID KRAMER

Definitive Guide to Excel VBA, Second Edition
Copyright ©2003 by Michael Kofler

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN (pbk): 1-59059-103-8

Printed and bound in the United States of America 12345678910

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Translator and Editor: David Kramer

Editorial Board: Dan Appleman, Craig Berry, Gary Cornell, Tony Davis, Steven Rycroft, Julian Skinner, Martin Streicher, Jim Sumser, Karen Watterson, Gavin Wright, John Zukowski

Assistant Publisher and Project Manager: Grace Wong

Copy Editor: Rebecca Rider

Production Manager: Kari Brooks

Production Editor: Janet Vail

Proofreader: Lori Bring

Compositor: Diana Van Winkle, vwdesign.com

Indexer: Kevin Broccoli

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, email orders@springer-ny.com, or visit <http://www.springer-ny.com>. Outside the United States: fax +49 6221 345229, email orders@springer.de, or visit <http://www.springer.de>.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit <http://www.apress.com>.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com> in the Downloads section.

Charts and Drawing Objects (Shapes)

CHARTS CONSTITUTE THE central feature of many Excel applications. This chapter gives a brief overview of the charts supported by Excel and also shows how you can create and print out your own charts under program control. A lengthy example on the subject of data recording demonstrates various programming techniques.

A further topic of this chapter is that of drawing objects (*Shapes*), available since Excel 97, with which both charts and ordinary worksheets can be ornamented.

Chapter Overview

10.1 <i>Charts</i>	528
10.2 <i>Programming Charts</i>	535
10.3 <i>Example: Automatic Data Reporting</i>	544
10.4 <i>Syntax Summary for Charts</i>	558
10.5 <i>Drawing Objects (Shapes)</i>	560
10.6 <i>Diagrams</i>	565

10.1 Charts

Fear not! You are not about to be subjected to an extensive introduction to the use of charts. This topic is exhaustively (in the literal sense of the word) dealt with in countless books on Excel. The goal of this section is rather to describe, without much concern about the details of how they are used, the possibilities for designing charts, and to name the various elements of charts and explain their functions. This information will provide you with the requisite knowledge for entering the world of programming charts, a world swarming with various *ChartXxx* objects.

Fundamentals

Chart Sheets Versus Charts Embedded in Worksheets

In Excel you can either embed charts in worksheets or present them in their own chart sheets. The first variant has the advantage that the chart can be printed out with its associated data. Furthermore, very small charts can be created that take up only part of a page.

The Chart Wizard

Usually, the path to a new chart goes by way of the chart wizard. This wizard is automatically summoned when you create a new chart (with `INSERT|CHART` or click on the chart wizard tool.)

In the first step of the chart wizard you select the desired chart type. In the second step you choose the data range. Here there is no problem with indicating a range of cells that is the union of other cell ranges. In further steps you can determine various options for the format of the chart. The chart wizard can also be called up to help with preexisting charts if you wish to change certain formatting details.

Further Processing of Charts

Charts that have been created with the chart wizard frequently do not quite meet your requirements. Therefore, the fine details of layout often begin after the chart wizard has been terminated.

In order for you to be able to edit the chart, you have to activate it with a mouse click. As soon as the chart is active, you can click on most of the chart elements within the region of the chart: the legend, the axes, individual data series

(which are represented in the form of lines, bars, etc.), the background of the chart, and so on. For each of these chart elements there exists a pop-up menu that usually offers an extensive array of formatting options. You can access the most important setting dialogs with a double click on the corresponding chart element.

If you are working with charts for the first time, you will often encounter the problem that you do not know which element to click on to carry out a specific change. You have two alternatives: Suffering through the user's guide and experimentation.

Chart Types

There are over seventy types of chart in Excel (though many of them are similar to one another). A complete list can be found in the chart wizard (where the charts are organized by group) or in VBA help under the keyword *ChartType*.

Combination Charts

Combination charts are charts in which several chart types are combined (for example, a line chart and column chart). Combination charts can be created either with the help of a user-defined chart (see below) or by changing the chart type of a single data series (not the entire chart).

Charts can be combined only if they are based on the same coordinate system. Therefore, the range of combination possibilities is relatively narrow. Three-dimensional charts cannot be combined at all.

Pivot Charts

Pivot charts are new in Excel 2000. These are not actually a new chart type, but a new way of linking data between a chart and a pivot table. What is special about pivot charts is that categories for structuring data can be created dynamically (that is, by means of listboxes in charts). The chart is immediately revised. For the chart itself almost all the chart types listed above can be used. Pivot charts will be described within the framework of pivot tables in Chapter 13.

User-Defined Chart Types (Autoformat)

There are two ways to format a chart: You can select one of the standard types, or you can employ a so-called user-defined type (formerly autoformat). Among these

types are stored numerous formatting details, so that you can very quickly create a wide variety of different charts. The name “user-defined” is somewhat confusing, since Excel recognizes an entire palette of predefined (integrated) types.

The user-defined formats in Excel give a very good overview as to what is available. The formats are located in `Officedirectory\Office\n\Xl8galry.xls`, where *n* is a language code (for example, 1033 is the number of the American version).

More important is the possibility of adding your own user-defined formats and using them in the future. For this you format a chart to your specifications, open the “Chart Type” dialog with the right mouse button, switch into the page “Custom Types,” and click on the option button “User-defined.” Then click the Add button and save your format as a new chart type. New (personal) chart types are saved in the file `Userprofile\Application Data\Microsoft\Excel\Xlusrgal.xls`.

Chart Elements (Chart Objects) and Formatting Options

For the detailed layout of charts as well as for programming charts it is necessary to know the distinctions made by Excel among various chart objects. Assistance in your experimentation is offered by the chart toolbar. There, in the left listbox is shown the object that was just clicked on, such as “axis *n*,” “gridlines *n*,” “Series *n*,” and so on.

- *Chart Area*: This is the object *ChartArea*, which is responsible for the background of the entire chart (that is, the region that is visible behind the plot area, the legend, and so on). The type style that is input here holds for all text of the chart that is not otherwise specially set.
- *Plot Area*: The plot area (*PlotArea*) represents a rectangle around the graphic region of the chart. The plot area contains the actual chart, but not the title, legend, etc. With most two-dimensional charts even the axes are not part of the plot area. If, for example, you specify the background color green for the plot area and red for the chart area, the labels for the axes will be underlaid with red.
- *Floor, Walls*: These two objects exist only for three-dimensional charts and describe the appearance of the floor and walls of the two vertical border surfaces of the chart. The plot area in this case is considered to be only the rectangular region outside of the chart itself.
- *Corners*: Even the corners exist as an independent object in three-dimensional charts. Corners cannot be formatted. But they can be grabbed with the mouse and turned in three dimensions. This is often more convenient than setting the viewpoint and perspective via the dialog `CHART|3D-VIEW`.

- *Data Series*: A data series describes a related unit of data (usually the values of a column from the underlying table; only if you select “Series in Rows” in step 2 of the chart wizard will data series be organized by rows). For example, a data series is represented by a line. The formatting data of data series affect the graphic representation of this data series, that is, color, markers, line style, etc.
- *Data Points*: The individual values of a data series are represented by data points. Normally, the format properties of all data points are the same and are preset by the properties of the data series. However, you can set the properties of each data point separately and thereby thrust individual points of a series into prominence, or label points individually, for example. In a pie chart you can shove individual pie slices out from the pie and distinguish them in this way—that, too, affects the property of the data point. Caution: The vertical position of data points in two-dimensional charts can be changed with the mouse, and this changes the underlying value in the data table!
- *Trend Lines*: Data series of two-dimensional charts can be associated with trend lines. The trend lines are drawn in addition to the normal representation of the data. Excel recognizes types of trend lines: best-fit curves (five different types) and averaging curves.
- *Error Bars*: Error bars are another subelement of a data series in two-dimensional charts. They indicate potential error amounts relative to each data marker.
- *Coordinate Axes*: The coordinate axes have a large number of formatting details, which begin with scaling (minimum, maximum, linear or logarithmic) and end with the precise arrangement of the axis labeling (which data points are labeled, which are indicated by a tick marks, whether the tick mark is inside or outside, and so on). New since Excel 97 is the possibility of labeling the coordinate axes with text in any orientation (horizontal, vertical, or slant; `FORMAT AXIS|FONT|ALIGNMENT`).

There is also the option to equip a two-dimensional chart with two independent Y-axes, where one is valid for some of the data series and the other for the remaining data series. This is useful when you wish to represent on the same chart two related quantities that have different scales (for example, a voltage and current). In order to employ two Y-axes it is necessary to separate the data series into two groups. The easiest way to accomplish this is by selecting the custom chart type “Lines on 2 Axes.”

- *Grid Lines*: The plot area of a two-dimensional chart or the walls and floor of a three-dimensional chart can be combined with grid lines. The position of

grid lines is determined by the tick marks on the coordinate axes. The appearance (color, line style) of principal and secondary grid lines can be set separately (but only for normal charts, not composite charts).

- *Title*: A chart can be equipped with several titles (for the chart, the axes, etc.). The position, type style, and alignment can be set independently.
- *Legend*: The legend makes possible a link between the colors used in the chart and the patterns of the data series. The labeling of the legend is taken from the first column or row of the data series. The legend can be placed anywhere in the chart (even beneath the data).

Chart Options in Tools|Options

With `TOOLS|OPTIONS|CHART` you have access to a few further chart options. These settings concern only the current chart (and can be changed only when the chart is active).

The option “Plot empty cells as” determines how Excel responds to empty cells in the data series. In the setting “not plotted (leave gaps)” there appears a hole in the chart (that is, bars are missing, a line is broken, etc.). The alternatives to this setting are “zero” (then Excel treats empty cells as if they contained the value 0) and “interpolated” (then Excel attempts to interpolate suitable data values for the empty cells).

The check box “Plot visible cells only” determines how Excel deals with hidden rows and columns: If the box is activated, then data in invisible rows or columns are not displayed. In the chart the data are simply ignored (rather than a hole appearing). This setting is of interest primarily when the chart data come from a filtered database.

The check box “Chart sizes with window frame” is of interest only for chart sheets. When the box is activated the chart is fit to the current size of the window. Otherwise, only one print page is shown. To make the entire chart visible, the zoom factor may have to be changed (`VIEW|ZOOM`).

Trend Lines, Data Smoothing

With line charts you can select the option “Smoothed line” in the formatting settings for the data series. This has the effect of rounding the edges of an otherwise angular course.

Other possibilities for providing a best-fit curve or averaging curve are offered by the command `CHART|ADD TRENDLINE`. Excel can approximate a data set with five different types of best-fit curves: straight line, polynomial curve (up to sixth

degree), logarithmic curve, exponential curve, power curve. With the options in the dialog **FORMAT TRENDLINE** you can specify whether and to what extent the curve should be extended beyond the current data and whether the formula for the curve should be given.

A sixth type of curve can be specified in the **TREND LINE** dialog: an averaging curve based on a running average. Here every point on the curve is calculated from the average of the n preceding points. This has the effect of smoothing statistical errors of measurement. Averaging curves, in contrast to best-fit curves, cannot be extended beyond the range of the data.

Some examples of the application of the trend line function are indicated in Figure 10-1. The associated example file **Trend.xls** is included with the sample files for this book.

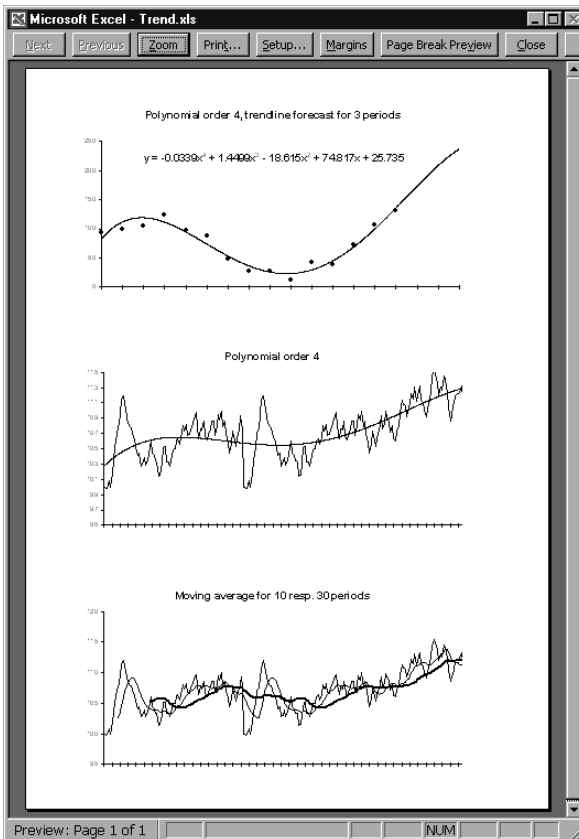


Figure 10-1. Three examples of trend lines

Error Indication

Data series in two-dimensional charts can be provided with error indicators (error bars). These are small lines that specify the range in which the actual value of a data point is to be found if statistical error of measurement is taken into account.

Printing

When it comes to printing a chart two variants must be considered: If the chart is embedded in a worksheet, then printing is accomplished by way of printing the worksheet. Here the only problem is that Excel does not give much thought to where page breaks are inserted, and even a small chart might find itself broken into four pieces. It does not hurt to check the page preview before printing. You may find yourself compelled to insert some hard page breaks to optimize printing (INSERT|PAGE BREAK).

On the other hand, if the chart is located in a chart sheet, or if you wish to print an embedded chart that has been selected with a mouse click, then there are some options available in the dialog PAGE SETUP. The most important of these is “Printed chart size” on the CHART page of this dialog. In the standard setting Excel uses the entire page. If the chart does not happen to have the same format as the page, the chart can become completely distorted. Therefore, it is usually better to select the option “Scale to fit page.” Excel enlarges the chart only to the extent that the relationship between the length and width does not change (that is, the aspect ratio is preserved). The third variant, “Custom,” leaves the size of the chart unchanged.

The option “Print in black and white” allows color charts to be printed on a black and white printer. (Most printers can handle this without the use of this option.) Whether with or without this option, you will achieve usable results on a black and white printer only if you refrain from using color in your chart. Use instead differing line widths and types to distinguish among several data series.

Since the standard and custom chart types are generally extremely color friendly, the creation of a satisfactory black and white substitute usually requires considerable effort (say, about 100 mouse clicks for a typical chart). Therefore, if this is a common situation for you, then save black and white charts as a custom chart type.

10.2 Programming Charts

First attempts at programming charts are often very difficult. The reason is that it is not easy to acquire an orientation among the multitude of *Chart* objects, and the association of properties and methods is not always clear.

Here is an example: The method *ClearContents* of the *ChartArea* object clears the data of a chart, but not its formatting. This is strange, in that the *ChartArea* object is actually not responsible for the chart itself, but only for its background. It would have been more logical if chart data were deleted via the *Delete* method of the *Chart* object, but this method returns nothing but an error message in the case of an embedded chart. Apparently, *Delete* is suitable only for deleting chart *sheets*, while the two related methods *ClearContents* and *ClearFormats* of the *ChartArea* object are responsible for the internal affairs of charts.

In contrast to the *ChartArea* object we have the *PlotArea* object. This object also describes the background of the chart, though in this case the area immediately behind the chart lines, bars, and so on.

REMARK *Though at the outset you may feel overwhelmed by the surfeit of objects and their properties, there are positive aspects to the situation: You can truly run almost the entire chart business with program code. Alas, space does permit a full description of this plenitude. For many details you will be referred to the on-line help after finishing this chapter.*

Instead of Searching Fruitlessly, Use the Macro Recorder!

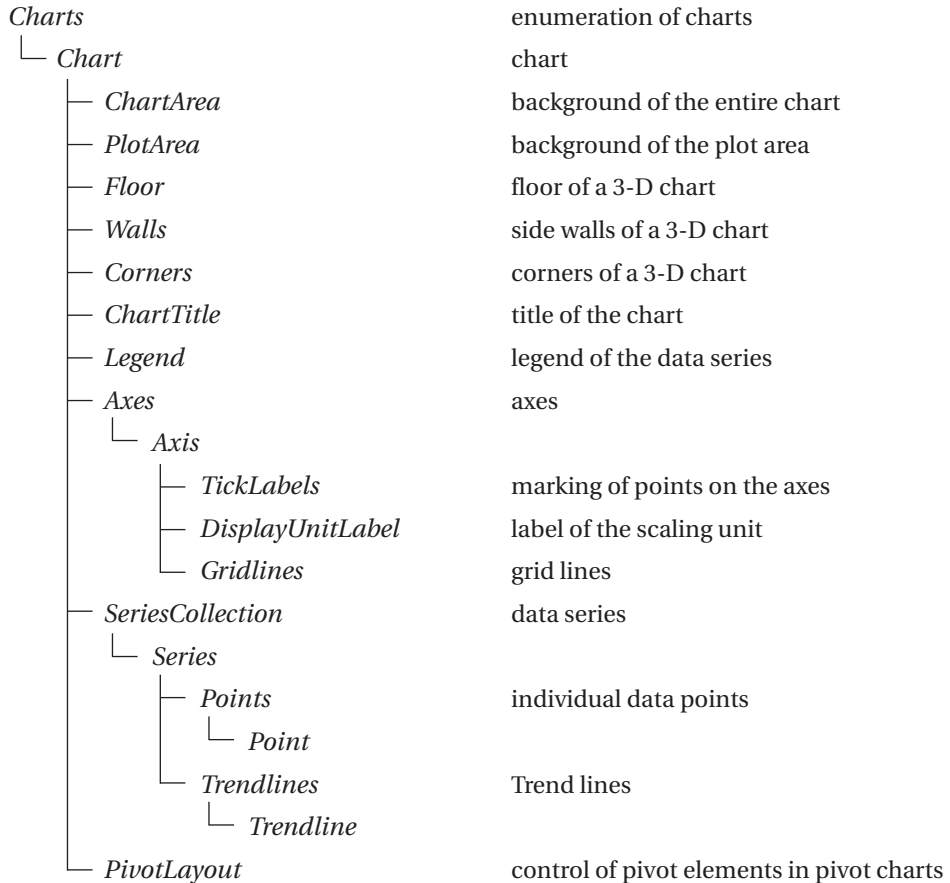
If you would like to know how you can achieve a particular formatting result in program code, then use the macro recorder as your trusted adviser (the examples from the on-line help are practically useless).

The shorter the recording session, the easier it will be for you to interpret the results. Therefore, you should start recording in a chart that already exists, change only a single detail, and then stop recording at once. If you arrange on your monitor one window with program code and a second one with the chart, you can even observe during recording when each line of code is generated.

The code that results from the macro recording usually works (at least no counterexamples appeared during the preparation of this chapter), but it is seldom optimal. In part, the instructions are unnecessarily convoluted, and in part they are completely superfluous. Therefore, the code must be edited after the fact.

Object Hierarchy

The following compilation provides an overview of the object hierarchy for charts. To make the structure clearer, only the most important objects have been included and only the case considered that the chart is embedded in a worksheet (no chart sheets). A complete listing of all chart objects can be found in Chapter 15.



A Brief Glossary of Chart Objects

There is enormous confusion surrounding the numerous and often like-named *Chart* and *Plot* objects. Figure 10-2 provides a first overview.

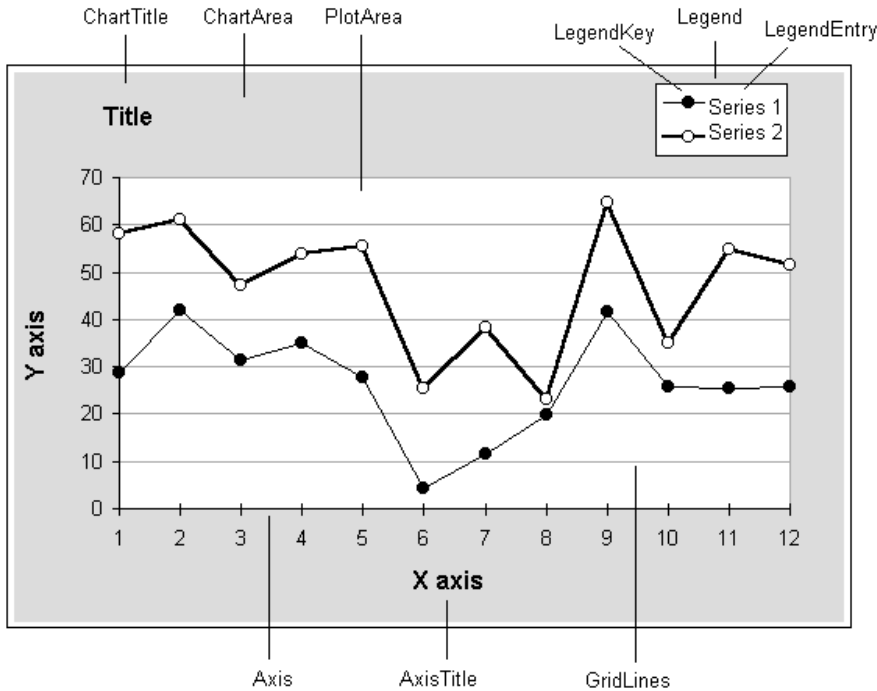


Figure 10-2. The most important objects of a chart

Chart: This is the actual chart; it consists of several data series that are graphically represented, the background, the coordinate axes, the legend, the title, and so on. Access to *Chart* objects is achieved either through the enumeration *Charts*, if the chart is located in a chart sheet, or via *ChartObjects(...).Chart*, if the chart is embedded in a worksheet.

The chart type is set, since Excel 97, with the property *ChartType* (formerly *Type* and *SubType*). Over seventy constants have been defined as possible settings (see the on-line help or the object browser).

ChartObject: This is the outer frame (container) of a chart. The *ChartObject* object is necessary only with charts that are embedded in worksheets. It stands between the worksheet and the *Chart* object and determines the position and dimensions of the chart within the worksheet. With the *Worksheet* method *ChartObjects* you can access the list of all chart objects of a worksheet.

You access the associated chart with the *Chart* property of the *ChartObject* object. (Note: In addition to charts a host of other objects can be embedded in worksheets, such as controls, lines, and rectangles. You can access the totality of these objects, including charts, with the method *DrawingObjects*).

ChartArea: This is the background of the chart. With the properties of this object you can set color, borders, and so on. However, this object has a greater significance insofar as its methods *Copy*, *Clear*, *ClearContents*, and *ClearFormats* relate to the actual subordinate *Chart* object (Microsoft alone knows why). In the case of embedded charts the method *Select* can be used only if first the associated *ChartObject* object has been activated with *Activate*.

ChartGroup: This object groups various chart types within a chart. Normally, a chart possesses only a single chart group. In this case the *ChartGroup* object is irrelevant. This object, then, has significance only when in a composite chart two or more chart types are united (for example, a bar chart and line chart). In this case the chart is managed by several groups with differing chart types (*Type* property).

Charts: The chart object contains the enumeration of all chart *sheets* of a workbook. The like-named method immediately returns the *Chart* object. There is, then, no separate chart sheet object comparable to a worksheet. For chart sheets, no intermediate *ChartObject* is necessary.

Some additional objects do not, in fact, begin with “Chart,” but they are nonetheless of interest.

PlotArea: This is the “graphical” area within a chart. The plot area contains the coordinate axes and the actual chart graphic. The main task of this object consists in determining the size and position of this region within the total area of the chart. Other regions in the chart are the legend (**Legend** object) and the title (**ChartTitle** object). In the case of three-dimensional charts the objects **Floor** and **Walls** (as subobjects of *Chart*) are managed independently of *PlotArea*. These two objects are responsible for the visual appearance of the boundary surfaces of a three-dimensional chart.

NOTE When you execute `PlotArea.Width=n; m=PlotArea.Width`, then *m* is distinctly larger than *n*. The reason is that `PlotArea.Width` actually changes the write-protected property `InsideWidth` introduced in Excel 97, that is, the inside region of `PlotArea`. In addition to this inside region there is an outside region, in which the labeling of the coordinate axes appears. (The same problems occur also with `Height/InsideHeight`, of course). To set the size of the outside region you can usually rely on the following code:

```
delta = PlotArea.Width - PlotArea.InsideWidth
PlotArea.Width = n + delta
```

This method is not quite exact either, since the size of the label area is not constant. For example, if a chart is greatly reduced in size, Excel simply does without axis labels, and the label area is reduced to size 0.

Series, Point: The *Series* object refers to the data of a data series belonging to a chart. The actual numerical values can be taken from the *Values* property of the *Series* object, which can also be used to change these values. *Series* is a subobject of the *Chart* object. Formatting data that affect not the entire series but only an individual data point are controlled by *Point* objects. These are again a subobject of the *Series* objects.

Axis, Gridlines: The *Axis* object is also a subobject of the *Chart* object. It describes the details of a coordinate axis. The *Gridlines* object is a subobject of the *Axis* object and is addressed via the properties *MajorGridlines* and *MinorGridlines*.

New in Excel 2000 is the ability to specify scaling units for the coordinate axes (for example, “millions”). In this case *Axis.DisplayUnit* must be set with a predefined constant (for example, *xlMillions*). The *DisplayUnitLabel* object specifies how and where this scale unit (that is, in this example “Millions”) is displayed in the chart. The property *HasDisplayUnitLabel* specifies whether the axis is scaled.

In addition to the predefined scaling units (10, 100, 1000, up to 1,000,000,000) any other factor can be used. For this, *DisplayUnitCustom* is assigned the desired value (which can also be less than 1, for example, 0.001 to represent thousandths). See Figure 10-3, in which the Y-axis is displayed in units of thousandths.

```
ActiveChart.Axes(xlValue).DisplayUnitCustom = 0.001
```

```
ActiveChart.Axes(xlValue).DisplayUnitLabel.Text = "thousandth part"
```

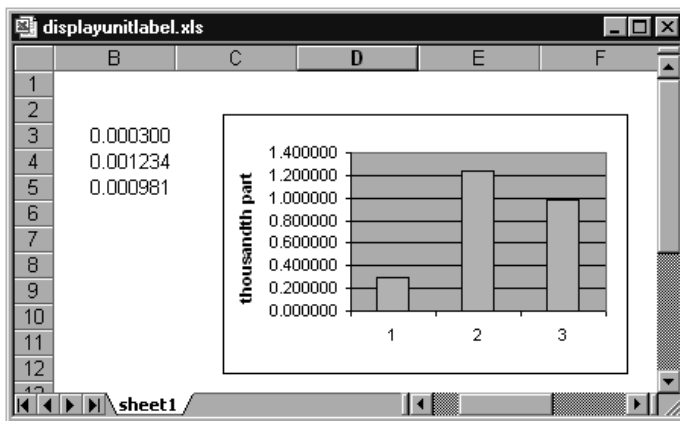


Figure 10-3. The Y-axis uses thousandths as scaling unit.

Trendline, ErrorBars: *Trendline* and *ErrorBars* are subobjects of the *Series* object. They describe the details of a trend line of a data series and, respectively, the appearance of the error bars.

TIP The keywords *Gridlines* and *ErrorBars* appear in the plural, and in contrast to other pluralized Excel objects, they do not refer to enumeration objects.

TIP If you do not know the name of a particular object in a chart, you can click on that object (for example, a coordinate axis) and execute `?TypeName(Selection)` in the immediate window. As result you obtain the object name (in this example, *Axis*).

Programming Techniques

The Chart Wizard

The method **ChartWizard** offers, in general, the fastest route to creating a chart. In order to use this method, you must first generate a *ChartObject* object. You can read all about the infinitude of parameters associated to this method in the on-line help.

```
ActiveSheet.ChartObjects.Add(30, 150, 400, 185).Name = _
    "new chart"
ActiveSheet.ChartObjects("new chart").Activate
ActiveChart.ChartWizard sheet1 .[A3:D99], xlLine, 4, xlColumns, 1, 1
```

Chart Objects: Activate or Select?

With the methods **Activate** and **Select** Microsoft has blessed us with a certain amount of confusion: Sometimes one method must be used (windows), sometimes the other (worksheets), and sometimes both are allowed (ranges of cells). In the case of *ChartObject* objects not only are both methods allowed, but they lead to different results!

Activate corresponds to a single click on a chart. *Selection* now refers to the object *PlotArea* (thus not to *Chart*)!

Select seemingly also corresponds to a single click on a chart. However, the *Selection* property now refers to a *ChartObject* object. Therefore, use *Select* when you wish to change the position or size of a chart in a worksheet but you do not wish to change features of the actual chart.

```
Sheets(1).ChartObjects(1).Select
```


The two methods have in common that the *Chart* object can then be accessed via *ActiveChart*.

CAUTION *In Excel 97 access to a chart via ActiveChart led at times to serious problems. Access the object in question directly instead of first activating it and then altering it via ActiveChart.*

Deactivating Chart Objects

The best way to deactivate a chart is by activating some other object. For example,

```
Sheets(n).[A1].Select
```

Deleting, Copying, and Inserting Charts

ChartObject objects can be directly copied together with the chart contained therein with **Copy** and then again inserted into the worksheet. After the insertion, the *Selection* property refers to the new *ChartObject* object, so that this can then be named. If you simply wish to duplicate a *ChartObject* object, you can use the method **Duplicate** directly instead of *Copy* and *Paste*. With *Delete* you can delete a *ChartObject* object together with all the data contained therein.

```
ActiveSheet.ChartObjects(1).Copy
ActiveSheet.Paste
Selection.Name = "new chart"
' ...
ActiveSheet.ChartObjects("new chart").Delete
```

The situation is somewhat different if you wish to delete, copy, or insert only the chart data without altering the *ChartObject* object. In this case the *ChartArea* object takes center stage (since there is no *Copy* method defined for the *Chart* object). Upon insertion into another chart object you must then refer to that *Chart* object.

```
ActiveSheet.ChartObjects(1).Chart.ChartArea.Copy
ActiveSheet.ChartObjects(2).Chart.Paste
```

With deleting chart data, too, you have to access the *ChartArea* object. *Clear* deletes all the chart data, *ClearContents* only the chart's contents (here is meant primarily the data series), and *ClearFormats* only the formatting information.

If you wish to insert an empty *ChartObject* into a worksheet (that is, an empty chart framework), you apply the *Add* method to *ChartObjects*. To this method are passed the position and size specifications (in points: 1 point = 1/72 inch = 0.35 mm). A name can be given at once to the new object:

```
ActiveSheet.ChartObjects.Add(0,0,200,100).Name = "new chart"
```

Aligning Several Charts

When you place two or more charts in a worksheet with the mouse, you will soon find out that it is relatively difficult to create two charts of exactly the same size lying one precisely above the other. A very good assistant in this enterprise is the menu of the DRAWING toolbar. With its menu items you can align previously selected objects (including charts). Another variant consists in simply accessing the *Left*, *Top*, *Width*, and *Height* properties of the *ChartObject* object.

The following instructions in the immediate window were used to align the five charts of a monthly report (see the next section) horizontally corresponding to the position and size of the first chart.

```
set wb = Worksheets("MonthlyReport")
For i=2 To 5: wb.ChartObjects(i).Left = _
    wb.ChartObjects(1).Left: Next i
For i=2 To 5: wb.ChartObjects(i).Width = _
    wb.ChartObjects(1).Width: Next i
```

Using Ready-Made Charts or Custom Formats

The complete setting up of a chart with all its formatting details is possible via program code, but this is a laborious and complex programming endeavor. If the appearance of a chart is in any case predetermined (and independent of the data to be processed) it makes more sense to save the completed chart in a worksheet or chart sheet and use program code only to change the data used to draw the chart. The actual formatting of the chart can be carried out directly with the mouse and without programming effort. (The procedure *MonthlyProtocol* in the next section provides an example.)

The use of autoformat requires more than minimal programming, but is still better than programming a chart from scratch. With autoformats, which in turn

are derived from charts that you have formatted in the traditional way, you can change practically all the formatting data of a chart generated by program code with a single instruction. Then you need to carry out at most a few instructions for the optimal sizing of individual chart elements. The application of an autoformat to an existing chart is carried out, since Excel 97, with the method ***ApplyCustomType*** (formerly ***AutoFormat***).

```
ActiveChart.ApplyCustomType ChartType:=xlUserDefined, _
    TypeName:="DailyReport"
```

The deployment of autoformats is problematic when you wish to install a complete Excel application on another computer. Personal autoformats are stored in the file `Userprofile\Application Data\Microsoft\Excel\Xlusrgal.xls`. This file cannot be copied to another computer, because you would thereby overwrite the autoformats of another user. Thus the transmission of autoformats in a file is impossible.

However, there is a way around this restriction. You should include in your application a worksheet in which you have embedded a simple example chart for each autoformat used. When the program is launched, activate these example charts one after the other and save their format information as autoformats on the computer on which the application is being run.

```
Application.AddChartAutoFormat Chart:=ActiveChart, _
    Name:= "new autoformat", Description:=""
```

Unfortunately, there is no way to determine which autoformats have already been defined. An object such as *AutoFormats* does not exist in the current version of Excel.

Printing and Exporting Charts

Printing a chart is carried out with the method ***PrintOut***, which can be applied to both *Chart* and *Workbook* objects. Since Excel 97 charts can also be exported into a graphics file in various formats with ***Export***.

```
ActiveChart.Export "test.gif", "GIF"
```

According to the on-line documentation, in the second parameter you can provide all of the graphics formats for which export filters have been installed. What filters exist, what they are called, and how the program can determine whether a particular filter is installed are not revealed in the documentation.

Therefore, protect your procedures for exporting with *On Error*. Experiments with *Export* have succeeded with the following formatting character strings:

“GIF”, “JPEG”, “TIF”, “TIFF”, “PNG”

On the other hand, *“BMP”* and *“WMF”*, that is, the two standard Microsoft formats for bitmaps and for simple vector graphics, are not supported. If you require charts in these formats, you can use the method *CopyPicture*, which copies the chart to the clipboard. Unfortunately, the exportation ends there. That is, Excel provides no method to save the contents of the clipboard to a file.

10.3 Example: Automatic Data Reporting

The file *Chart.xls* demonstrates the application of Excel to the reporting of measurement data. Data reporting is necessary whenever relatively large data sets need to be documented and perhaps analyzed over an extended period of time. The data source can be just about anything, from the automatically measured amounts of hazardous chemicals in a waste treatment plant to the results of quality control in a factory.

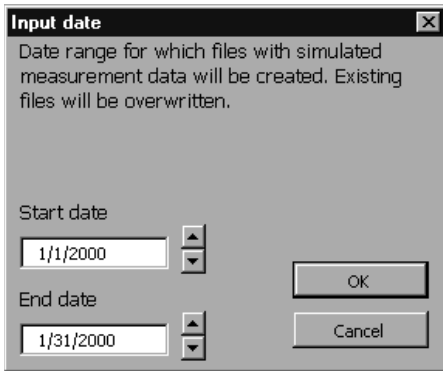
The task of data reporting is to generate informative and readable printouts from the trash heap of numbers consisting of many small, or one large, file or database. It should be clear that charts for data visualization can play an important role in this operation.

Since in the sample files we cannot provide a technological method of data production, the application *Chart.xls* has available the menu command *REPORT|CREATE TEST DATA*, which creates Excel files with simulated measurement data. In practice, you would need such a command only during the test phase of the program. In general, you would have more genuine measurement data at your disposal than you probably want, and you would not need to increase your data supply with a data simulation program.

Using the Example Program

When the file is opened, a custom menu appears. If you wish to try out the program quickly, execute the following commands in sequence: *REPORT|CREATE TEST DATA*, *|CREATE DAILY REPORT*, and *|CREATE MONTHLY REPORT*. Simply approve with “Ok” the forms (such as that shown in Figure 10-4) that appear for data input.

The program then produces for each day of the current month a data file (requiring about 900 kilobytes of storage and half a minute to create on a reasonably modern computer). Then the daily report for the current day and the monthly report for the current month are presented in page view.



Input date

Date range for which files with simulated measurement data will be created. Existing files will be overwritten.

Start date
1/1/2000

End date
1/31/2000

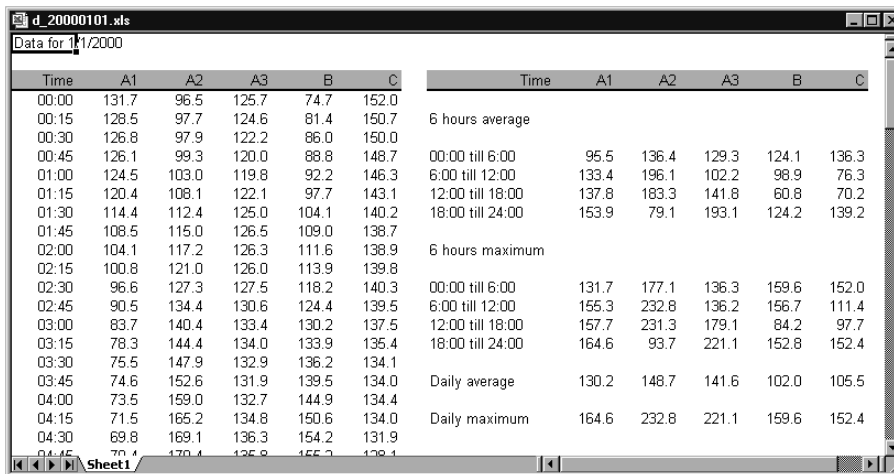
OK

Cancel

Figure 10-4. The form for input of the data range

Test Data

The menu command REPORT|CREATE TEST DATA leads to the creation, for each day, of files with the names `D_yyyymmdd.xls` (such as `D_19991231.xls` for 12/31/1999). In addition to the actual measurement data (96 values in each of the data series A1, A2, A3, B, C) these files contain six-hour average and maximum values as well as the daily average and maximum (see Figure 10-5). The files `D_yyyymmdd.xls` can be deleted after the program has been tested, of course.



`d_20000101.xls`
Data for 1/1/2000

Time	A1	A2	A3	B	C	Time	A1	A2	A3	B	C
00:00	131.7	96.5	125.7	74.7	152.0						
00:15	128.5	97.7	124.6	81.4	150.7						
00:30	126.8	97.9	122.2	86.0	150.0						
00:45	126.1	99.3	120.0	88.8	148.7	00:00 till 6:00	95.5	136.4	129.3	124.1	136.3
01:00	124.5	103.0	119.8	92.2	146.3	6:00 till 12:00	133.4	196.1	102.2	98.9	76.3
01:15	120.4	108.1	122.1	97.7	143.1	12:00 till 18:00	137.8	183.3	141.8	60.8	70.2
01:30	114.4	112.4	125.0	104.1	140.2	18:00 till 24:00	153.9	79.1	193.1	124.2	139.2
01:45	108.5	115.0	126.5	109.0	138.7						
02:00	104.1	117.2	126.3	111.6	138.9	6 hours maximum					
02:15	100.8	121.0	126.0	113.9	139.8						
02:30	96.6	127.3	127.5	118.2	140.3	00:00 till 6:00	131.7	177.1	136.3	159.6	152.0
02:45	90.5	134.4	130.6	124.4	139.5	6:00 till 12:00	155.3	232.8	136.2	156.7	111.4
03:00	83.7	140.4	133.4	130.2	137.5	12:00 till 18:00	157.7	231.3	179.1	84.2	97.7
03:15	78.3	144.4	134.0	133.9	135.4	18:00 till 24:00	164.6	93.7	221.1	152.8	152.4
03:30	75.5	147.9	132.9	136.2	134.1						
03:45	74.6	152.6	131.9	139.5	134.0	Daily average	130.2	148.7	141.6	102.0	105.5
04:00	73.5	159.0	132.7	144.9	134.4	Daily maximum	164.6	232.8	221.1	159.6	152.4
04:15	71.5	165.2	134.8	150.6	134.0						
04:30	69.8	169.1	136.3	154.2	131.9						

Figure 10-5. The construction of the daily files for the measurement data

In the reporting of the data it is assumed that the dataseries A1, A2, A3 are related. Therefore, these series are presented in a single chart (see Figure 10-6). In the monthly report this was no longer possible due to the complexity of the data, since in the charts for each data series the daily average as well as the daily maximum are presented in their own graph (see Figure 10-7).

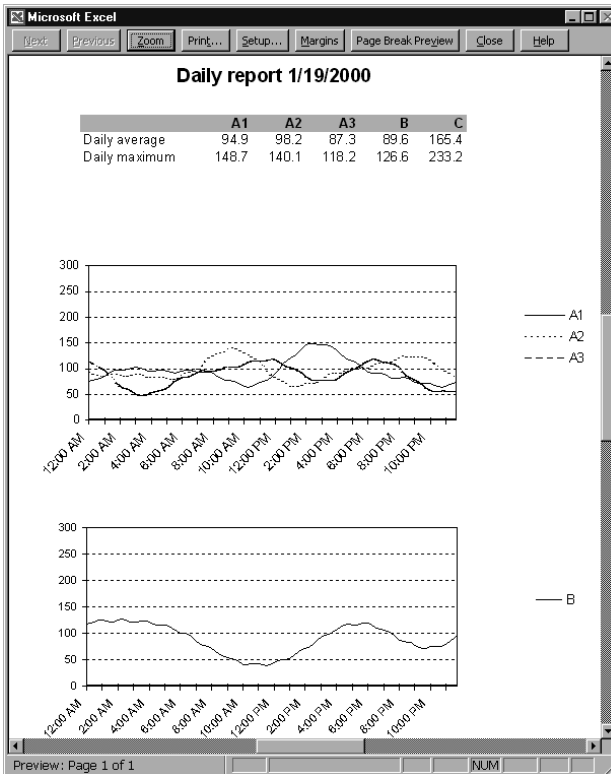


Figure 10-6. A daily report

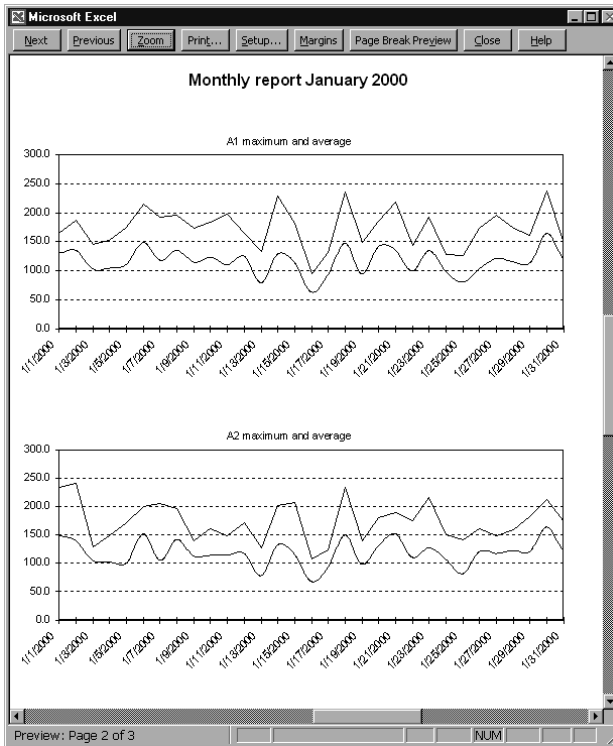


Figure 10-7. A page from the three-page monthly report

Program Code

Overview of the Components of Chart.xls

The Excel file `Chart.xls` consists of the following worksheets:

- “Intro”: Worksheet with information about the use of the application.
- “DailyReport”: Worksheet in which the daily report is constructed. The charts contained in it are deleted for each new report and constructed anew.
- “MonthlyReport”: Worksheet in which the monthly report is constructed. The charts contained in it are final; they are not changed further in program code. In program code only the content of cells B9:M39 is changed.
- “DataTemplate”: Worksheet that serves as template for the files with simulated data.

The construction of the worksheets must not be altered, since access to particular cells is carried out directly in program code.

The program code is divided into the following modules:

“ <i>ThisWorkbook</i> ”:	display menu on opening; delete it on closing.
“ <i>FormDateInput</i> ”:	form for input of date range.
“ <i>MenuEvents</i> ”:	event procedures for the menu commands.
“ <i>CreateDateFiles</i> ”:	procedures for generating the test data.
“ <i>CreateReports</i> ”:	procedures for building and printing the daily and monthly reports.

On the following pages the most interesting details of the program code are described. The same order is observed as that for using the program (generate test data, daily report, monthly report). The code not only demonstrates the various possibilities for chart programming, it also shows how you can consolidate data from several Excel files when the Excel function `DATACONSOLIDATE` is too inflexible for your requirements.

Creating the Test Data

The program segment for creating the test data is of little interest to the extent that it would not exist in a real-world application (in which one has genuine data!). In our example *GenerateDailyWorksheet* creates a new Excel file based on the template worksheet in the sheet “DataTemplate.” This template contains not only various formatting data, but also some formulas for calculating the six-hour average and maximum values as well as the daily average and maximum values.

The simulated test data are calculated on the basis of six superposed sine curves of various frequencies. The parameters of these functions (amplitude, frequency, and phase) are stored in the global field *rndmat*. The global variable *rndInit* determines whether this field already contains valid values. This avoids the necessity of providing new random numbers for each day. (Random numbers are generated only the first time this procedure is called.)

The random numbers are initialized in the procedure *InitRandomnumbers* (not presented here). Here the attempt is made to choose similar values for the three data series A1, A2, A3. For each day the procedure *DailyRandomnumbers* is called anew. This procedure changes the existing values of the *zfmt* field by a small amount, so that the data do not appear to be too regular.


```

' Chart.xls, Module CreateDataFiles
Dim rndInit As Boolean          'tests whether random matrix is already initialized
Dim rndmat#(5, 18)             'matrix with random numbers
Const Pi = 3.1415927
' create workbook with (random) measurement data for one day
Function GenerateDailyWorksheet(dat As Date) As Boolean
    Dim filename$              'name of the new workbook
    Dim wb As Workbook         'new workbook
    Dim ws As Worksheet        'sheet in this book
    Dim cell As Range          'first data cell on the sheet
    Dim i%, j%, k%             'loop variables
    Dim x#, z As Date

    filename = ThisWorkbook.Path + "\d_" + _
        Format(dat, "yyyymmdd") + ".xls"
    Application.DisplayAlerts = False
    ' creates new workbook; copies sheet "DataTemplate" from
    ' this workbook into new workbook; deletes all other sheets
    Set wb = Workbooks.Add
    ThisWorkbook.Sheets("DataTemplate").Copy Before:=wb.Sheets(1)
    For i = wb.Sheets.Count To 2 Step -1
        wb.Sheets(i).Delete
    Next i
    wb.Sheets(1).Name = "Sheet1"
    ' insert random numbers into sheet
    Set ws = wb.Worksheets(1)
    Set cell = ws.[A4]
    ws.[a1] = "Data for " & dat
    If Not rndInit Then InitRandomnumbers
    DailyRandomnumbers
    Application.Calculation = xlManual
    For i = 1 To 96                '00:00 through 23:45
        z = dat + CDBl(#12:15:00 AM#) * (i - 1)
        cell.Cells(i, 1) = z
        cell.Cells(i, 1).NumberFormat = "hh:mm"
        For j = 1 To 5            'five series of data
            x = rndmat(j, 0)
            For k = 1 To 18 Step 3
                x = x + rndmat(j, k) * (1 + Sin(rndmat(j, k + 1) * z + _
                    rndmat(j, k + 2)))
            Next k
            cell.Cells(i, j + 1) = x
        Next j
    Next i
End Function

```

```

Next i
Application.Calculation = xlAutomatic
Application.DisplayAlerts = True
On Error Resume Next
' delete existing file
If Dir(filename)<>" " Then Kill filename
wb.SaveAs filename
wb.Close False
If Err = 0 Then
    GenerateDailyWorksheet = True
Else
    MsgBox "An error has occurred: " & Error
    GenerateDailyWorksheet = False
End If
End Function

```

NOTE *It happens again and again with automated measuring processes that due to some error, data are missing for a period of time (hours, or even days). In the procedure above, error simulation was not implemented. However, the reporting in DailyReport and MonthlyReport will continue to function if you simply delete some of the data from the generated files. But be careful in the calculation of average values. Missing measurements must not be taken to be zero values. The Excel worksheet function AVERAGE behaves admirably in this case and considers only those cells in the given range that are not empty. Only when all of the measurements of an averaging range are missing does it return the error result “division by 0.”*

Daily Report

The daily report contains three charts, in which the exact course of the measurements is presented. Here the curves A1, A2, A3 are united in a single chart. So that charts from several days can be compared easily, a uniform scaling is required. For this reason the Y range is set with a fixed range of 0 to 300. (Normally, Excel changes the scaling automatically and fits it to the values that actually occur.) Integrated into the daily report are a tabular overview of the daily average values and the daily maximum of the five curves.

The daily report for a given date is created by the procedure *DailyProtocol*. The charts are created completely in program code and inserted into the worksheet “DailyReport.” Any existing charts in this worksheet (from the previous report) are first deleted.

The procedure opens the file with the daily data and copies some basic information (daily average and maximum) from it into the worksheet "DailyReport." Furthermore, the title of the report is extended to include the relevant date.

To generate a new chart, first three empty *ChartObject* frames are placed in the worksheet. Then *ChartWizard* is used to create charts within them corresponding for the most part to the actual requirements. (Some details that are not within control of *ChartWizard* have to be changed later on.) The three *ChartWizard* instructions differ only in that the charts are associated to differing ranges of cells from those of the daily data table.

Then begins the actual detail work of formatting the chart. The three charts can be worked on as a unit in a loop. The procedure ends with the daily data file being closed and the daily report being printed. (On account of the option *Preview:=True* printing takes the form of a page view.)

```
' Chart.xls, Module CreateReports
Sub DailyProtocol(dat As Date)
    Dim filename$           'report file name
    Dim protWBook As Workbook 'workbook of this file
    Dim protWSheet As Worksheet 'sheet of this book
    Dim protRange As Range   'first data cell in this sheet
    Dim chartWSheet As Worksheet 'reference to sheet with daily data
    Dim i%, chobj As ChartObject 'loop variables
    Application.ScreenUpdating = False
    filename = ThisWorkbook.Path + "\d_" + _
        Format(dat, "yyyymmdd") + ".xls"
    If Dir(filename) = "" Then
        MsgBox "The file " & filename & " does not exist. " & _
            "Please create test data."
        Exit Sub
    End If
    Set protWBook = Workbooks.Open(filename)
    Set protWSheet = protWBook.Worksheets(1)
    Set protRange = protWSheet.[A4]
    Set chartWSheet = ThisWorkbook.Worksheets("DailyReport")
    ' delete all existing charts on this sheet
    For Each chobj In chartWSheet.ChartObjects
        chobj.Delete
    Next chobj
    ' copy caption, daily averages and daily maximum values in table
    chartWSheet.[ReportLabel] = "Daily report " & dat
    protWSheet.[I19:M19].Copy
    chartWSheet.[DailyAverage].PasteSpecial xlValues
    protWSheet.[I21:M21].Copy
```

```

chartWSheet.[DailyMax].PasteSpecial xlValues
' create three charts
For i = 1 To 3
  chartWSheet.ChartObjects.Add(30, 150 + 200 * (i - 1), 400, 185). _
    Name = "Daily data " & i
  chartWSheet.ChartObjects("Daily data " & i).Activate
  If i = 1 Then
    ActiveChart.ChartWizard protWSheet.[A3:D99], _
      xlLine, 4, xlColumns, 1, 1
  ElseIf i = 2 Then
    ActiveChart.ChartWizard protWSheet.[A3:A99,E3:E99], _
      xlLine, 4, xlColumns, 1, 1
  ElseIf i = 3 Then
    ActiveChart.ChartWizard protWSheet.[A3:A99,F3:F99], _
      xlLine, 4, xlColumns, 1, 1
  End If
Next i
' format charts
For Each chobj In chartWSheet.ChartObjects
  chobj.Border.LineStyle = xlNone 'no border for entire chart
  With chobj.Chart
    .HasTitle = False 'no title
    .PlotArea.Border.LineStyle = xlAutomatic 'border
    .PlotArea.Interior.ColorIndex = xlNone 'no pattern/fill
    .Axes(xlCategory).TickLabelSpacing = 8
    .Axes(xlCategory).TickMarkSpacing = 4 'x axis
    .Axes(xlValue).MinimumScale = 0 'y axis
    .Axes(xlValue).MaximumScale = 300
    .Axes(xlCategory).TickLabels.Orientation = 45 '45 degrees
    .Axes(xlCategory).TickLabels.NumberFormat = "h:mm AM/PM"
    For i = 1 To .SeriesCollection.Count 'format data
      .SeriesCollection(i).Border.ColorIndex = 1 ' series
      .SeriesCollection(i).Border.Weight = xlThin
      .SeriesCollection(i).Border.LineStyle = xlContinuous
      .SeriesCollection(i).MarkerStyle = xlNone
    Next i
    If .SeriesCollection.Count > 2 Then 'distinguish
      .SeriesCollection(2).Border.LineStyle = xlDot ' 2nd and 3rd
      .SeriesCollection(3).Border.LineStyle = xlDash ' series
    End If
    ' diagram size, legend size
    .PlotArea.Left = 5: .PlotArea.Top = 5
    .PlotArea.Width = 290
  End With
Next chobj

```

```

        .PlotArea.Height = 140
        .Legend.Left = 340
        .Legend.Width = 50
        .Legend.Border.LineStyle = xlNone
    End With
Next chobj
ActiveWindow.Visible = False 'deactivate chart
protWBook.Close
chartWSheet.PrintOut Preview:=True
End Sub

```

Monthly Report

The monthly reports are somewhat more lavishly decked out than the daily reports, taking three pages in all. The first side consists of an overview of all daily average and maximum values as well as the resulting monthly averages and maxima. The next page contains three charts, and the last page sports two charts. These show the progression of the average and maximum values. The curves for the averages have been smoothed (click on the curve, open the pop-up menu `FORMAT DATA SERIES|PATTERNS`, option `SMOOTHED LINE`). Figure 10-7 shows the second page of the monthly report with the curves for the measurement values A1 through A3.

For generating the monthly report we have chosen a method completely different from that used for the daily report. The charts were inserted (with the mouse) into the worksheet “MonthlyReport” and are not touched at all by the procedure *MonthlyProtocol*. *MonthlyProtocol* merely changes those data cells that the finished chart accesses.

This way of proceeding has advantages and disadvantages. The advantage is that the programming effort is greatly reduced. Thus you can achieve good results with minimal experience in programming charts. The disadvantages become evident when you attempt to generate five identical charts by mouse click. This is almost as much effort as the programming (even if you first create a chart, and then copy it and change only the ranges of cells of the data series). Furthermore, this way of proceeding is possible only if the chart, as in this example, is to a great extent independent of the data. However, if such items as the number of data series, the number of data points, and the range of values of the data series can vary, then there is no avoiding “real” programming.

REMARKS *The charts assume a month of 31 days. In the case of months with fewer days there are one to three empty data points. Thus the space available for the chart is not used to full capacity, but in exchange there is a distinct advantage: The scaling of the X-axis is independent of the number of days in the month. The charts are thereby more comparable.*

Onward to the program code, which for the reasons cited above contains not a single line of instructions that typically apply to charts. The procedure is rather an example of how data from up to 31 files can be consolidated in a single table. The individual files are not opened, but rather direct access to individual cells of other worksheets is made via formulas of type =C:\Test\[D_20000101.XLS]Sheet1'!\$L\$19. This form of data access proceeds surprisingly quickly. The creation of the monthly report takes only a little longer than that of the daily report.

The most complicated part of the procedure relates to the creation of these formulas, which are inserted into the worksheet by changing the *FormulaR1C1* property of the affected cells. The formulas must be created relatively laboriously as character strings. The R1C1 format is better suited for such tasks, because at least there is no transformation from column numbers into letters.

```
Sub MonthlyProtocol(dat As Date)
    Dim sdat As Date, edat As Date 'start and end date
    Dim nrdays As Integer         'number of days
    Dim filename$                 'name of report file
    Dim chartWSheet As Worksheet 'sheet of report file
    Dim chartRange As Range       'first data cell
    Dim z As Date, i%, j%         'loop variables
    sdat = DateSerial(Year(dat), Month(dat), 1)
    nrdays = DateSerial(Year(dat), Month(dat) + 1, 1) - _
        DateSerial(Year(dat), Month(dat), 1)
    edat = dat + nrdays - 1
    ThisWorkbook.Activate
    Set chartWSheet = ThisWorkbook.Worksheets("MonthlyReport")
    chartWSheet.Activate
    chartWSheet.[a1].Select
    Set chartRange = chartWSheet.[B9]
    ' build monthly table
    Application.Calculation = xlManual
    chartWSheet.[B1] = "Monthly report " & Format(dat, "mmm yyyy")
    For i = 1 To nrdays
        z = dat + i - 1
```

```

chartRange.Cells(i, 1) = z
filename = ThisWorkbook.Path + "\d_" + _
          Format(z, "yyyymmdd") & ".xls"
If Dir(filename) = "" Then
    For j = 1 To 5
        chartRange.Cells(i, 1 + j).FormulaR1C1 = ""
        chartRange.Cells(i, 7 + j).FormulaR1C1 = ""
    Next j
Else
    filename = "=" & ThisWorkbook.Path + "\[d_" + _
              Format(z, "yyyymmdd") & ".xls]Sheet1'"
    For j = 1 To 5
        chartRange.Cells(i, 1 + j).FormulaR1C1 = _
            filename & "!R19C" & 8 + j
        chartRange.Cells(i, 7 + j).FormulaR1C1 = _
            filename & "!R21C" & 8 + j
    Next j
End If
Next i
If nrdays < 31 Then
    For i = nrdays + 1 To 31
        For j = 1 To 12
            chartRange.Cells(i, j).ClearContents
        Next j
    Next i
End If
Application.Calculate
chartWSheet.Range("B9:M39").Copy
chartWSheet.Range("B9:M39").PasteSpecial Paste:=xlValues
Application.CutCopyMode = False
chartWSheet.PrintOut Preview:=True
Application.Calculation = xlAutomatic
End Sub

```

After all references have been inserted into the worksheet and the worksheet recalculated, the entire range of cells is copied to the clipboard. Then, with *PasteSpecial* only the numerical values (instead of the formulas) are pasted. This process saves memory and increases the speed of further processing. Furthermore, it does not occur to Excel to ask at the next opportunity whether it should update the existing references.

The procedure ends, like *DailyProtocol*, with printing the worksheet together with the five charts contained therein. Furthermore, in the page layout of the worksheet (FILE|PAGE SETUP) “none” is selected for the header, and for the footer the page number is inserted (since the report always contains three pages).

Menu Management

The management of the menus has nothing new about it in comparison to what has been discussed in earlier chapters, for this reason we have not included the code here for the event procedures. The menu is realized as an independent *CommandBar* object. It is made visible in *Workbook_Open* when *Chart.xls* is opened, and is hidden again in *Workbook_BeforeClose*.

Dialog Management

The form *FormDateInput* is used universally for the three commands REPORT|CREATE TEST DATA, ...|DAILY REPORT, and ...|MONTHLY REPORT. The text in the text box *lblInfo* is changed according to the purpose for which it is to be used. With the procedures *ProtocolMenu_GenerateNewFiles*, *_DailyProtocol*, and *_MonthlyProtocol*, of which only one is reproduced here, the text in the text boxes *txtFrom* and *txtTo* is preset.

The two dates can be increased or decreased with spin buttons. The values are preset to 0, and the permissible range is from -1000 to 1000. Therefore, you can theoretically change the date by ± 1000 days. (Theoretically, because you would not have the patience to keep pushing the button. Much quicker is simply to input the date via the keyboard.)

```
' Chart.xls, Module MenuEvents
Sub ChartSampleMenu_MonthlyProtocol()
    Dim dat As Date, lastmonth As Integer
    lastmonth = -1
    With FormDateInput
        .dat1 = DateSerial(Year(Now), Month(Now), 1)
        .dat2 = DateSerial(Year(Now), Month(Now), _
            DateSerial(Year(Now), Month(Now) + 1, 1) - _
            DateSerial(Year(Now), Month(Now), 1))
        .txtFrom = CStr(.dat1)
        .txtTo = CStr(.dat2)
        .spinTo = 0
        .spinFrom = 0
    End With
End Sub
```



```

.lblInfo = "Date range for which monthly reports will be " & _
    "created and printed."
.Show
If .result = False Then Exit Sub
' create report
Application.ScreenUpdating = False
Application.DisplayStatusBar = True
For dat = CDate(.txtFrom) To CDate(.txtTo)
    If lastmonth <> Month(dat) Then
        Application.StatusBar = "Create monthly report for " & _
            Format(dat, "mmm yyyy")
        MonthlyProtocol CDate(dat)
        lastmonth = Month(dat)
    End If
Next dat
Application.StatusBar = False
Application.DisplayStatusBar = False
End With
End Sub

```

If the input is terminated with Ok and if in *btnOK_Click* no input error is discovered, then a loop runs through all the days of the date range. Each time the month changes, *MonthlyProtocol* is called. Admittedly, the algorithm has not been overly carefully programmed, but it is surely the simplest solution that functions for arbitrary time intervals (even for more than twelve months). A calculation of the first day of each new month would probably require more time than simply running through all the days. In any case, it would have required more thought in the programming, and programmers are known not always to be in the mood for heavy-duty thinking.

The actual form event procedures turn out to be comparatively short and trivial. Note that the spin button is not synchronized when a new date is input via the keyboard. For this reason it is impossible to input a date via the keyboard and then change it with the spin button.

```

' event procedure for the form for date input
Option Explicit
Public result As Boolean, dat1 As Date, dat2 As Date
Private Sub btnCancel_Click()
    result = False
    Hide
End Sub
Private Sub btnOK_Click()
    If IsDate(txtFrom) And IsDate(txtTo) Then

```

```

        result = True
    Hide
Else
    MsgBox "Invalid date!!"
End If
End Sub
Private Sub spinFrom_Change()
    txtFrom = CStr(dat1 + spinFrom)
End Sub
Private Sub spinTo_Change()
    txtTo = CStr(dat2 + spinTo)
End Sub

```

10.4 Syntax Summary for Charts

This section collects almost all the truly important chart objects, methods, and properties. A summary of the object hierarchy of all chart objects appears in Chapter 15. There, all objects are also briefly described. In the following syntax boxes we have used the following abbreviations: *wb* for a *Workbook* object, *ws* for a *Worksheet* object, *chobj* for a *ChartObject* object, and *ch* for a *Chart* object.

CHART OBJECTS

<i>ws.ChartObjects</i> (..)	select embedded chart object
<i>ws.ChartObjects.Add</i> ..	new (empty) chart frame
<i>chobj.Select</i>	corresponds to a single mouse click
<i>chobj.Activate</i>	corresponds to a single mouse click
<i>ActiveWindow.Visible = False</i>	deactivate
<i>chobj.Chart</i>	refers to a chart object
<i>chobj.Copy</i>	copy chart object together with chart
<i>ws.Paste.Selection.Name = ".."</i>	insert chart object together with chart
<i>chobj.Duplicate.Name = ".."</i>	duplicate existing chart object
<i>chobj.Delete</i>	delete chart object together with chart

CHARTS

<i>ActiveChart</i>	refers to the active chart
<i>wb.Charts(..).Select</i>	selects chart sheet
<i>ch.ChartArea.Copy</i>	copies chart contents
<i>ch.Paste</i>	inserts chart contents
<i>ch.ChartArea.Clear</i>	deletes entire chart
<i>ch.ChartArea.ClearContents</i>	deletes only the data
<i>ch.ChartArea.ClearFormats</i>	deletes only the format
<i>ch.ChartWizard ...</i>	create chart with chart wizard
<i>ch.ApplyCustomType ...</i>	use custom format
<i>Application.AddChartAutoFormat ...</i>	save new custom format
<i>ch.CopyPicture</i>	copies chart as graphic or bitmap to the clipboard
<i>ch.Export</i>	saves chart in a graphics file
<i>ch.PrintOut</i>	prints the chart
<i>ch.ChartArea</i>	refers to entire background
<i>ch.PlotArea</i>	refers to background of the graphic
<i>ch.Floor; ch.Walls</i>	refers to floor and walls (3-D chart)
<i>ch.ChartTitle</i>	refers to chart title
<i>ch.Legend</i>	refers to legend
<i>ch.Axes(..)</i>	refers to axes
<i>ch.SeriesCollection(..)</i>	refers to data series

10.5 Drawing Objects (*Shapes*)

Overview

The **Shape** object serves primarily to represent autoshapes (lines, rectangles, arrows, stars, etc.; see the “Drawing” toolbar).

These objects take the place of the various drawing objects in Excel 5/7. However, the large number of related objects can be a source of confusion.

HIERARCHY OF SHAPE OBJECTS

Worksheet/Chart

└─ <i>Shapes</i>	all <i>Shape</i> objects within a sheet
└─ <i>Shape</i>	a <i>Shape</i> object
├─ <i>ConnectorFormat</i>	connection to other objects
├─ <i>ControlFormat</i>	additional properties for controls
├─ <i>FillFormat</i>	background pattern (via <i>Fill</i> property)
├─ <i>GroupShapes</i>	single object (via <i>GroupItems</i> , if <i>Type</i> =msoGroup)
└─ <i>Shape</i>	
├─ <i>HyperLink</i>	cross link and internet links
├─ <i>LineFormat</i>	line properties (via <i>Line</i>)
├─ <i>LinkFormat</i>	additional properties for OLE objects
├─ <i>OLEFormat</i>	yet more properties for OLE objects
├─ <i>PictureFormat</i>	properties of picture objects
├─ <i>Range</i>	anchor cells (via <i>TopLeft-/BottomRightCell</i>)
├─ <i>Shadow</i>	properties for shadow
├─ <i>ShapeNodes</i>	line segment (via <i>Nodes</i> , if <i>Type</i> =msoFreeform)
└─ <i>ShapeNode</i>	
├─ <i>ShapeRange</i>	single objects with multiple editing (via <i>Range</i>)
└─ <i>Shape</i>	
├─ <i>TextEffectFormat</i>	properties for WordArt object
├─ <i>TextFrame</i>	Text box within an autoshape object
└─ <i>ThreeDFormat</i>	3-D effects (via <i>ThreeD</i>)

The *Shapes* enumeration enables access to all *Shape* objects of a worksheet or chart sheet. For the insertion of new drawing objects there is a long list of methods available, such as *AddShape* for autoshapes and *AddLine* for lines.

ShapeRange enables the simultaneous editing of several *Shape* objects (as if these objects were selected with Shift and the mouse).

Freehand shapes (that is, freely drawn line segments) represent a particular form of *Shape* objects. In this case, the property *ShapeNodes* refers to a like-named enumeration of **ShapeNode** objects. These objects contain, among other attributes, coordinate points of the individual line segments.

A *Shape* object is also used for managing a so-called group (in interactive mode: pop-up menu command `GROUPING`). In this case the property *GroupItems* leads to a **GroupShape** object, which, in turn, takes over the management of the group elements. Group elements can include not only *Shape* objects, but also charts and OLE objects, among others.

Finally, *Shape* is used to manage completely foreign objects, such as for MS Forms control objects (*Type=msoOLEControlObject*). In this case, *Shape* stands between the worksheet or chart sheet and the actual object. *Shape* is then concerned, among other things, with the positioning of the control. For communication between the sheet and the control the **ControlFormat** object is employed, which is addressed via the like-named property of *Shape*. *ControlFormat* is generally transparent, because its properties appear in the properties window of the control and can be used like control properties.

Shape Properties

AutoShapeType: The two most important properties are surely *Type* and *AutoShapeType*. If *Type=msoAutoShape* is set, then with *AutoShapeType* one of countless autoshape types can be specified (there are more than 130). On the other hand, if no autoshape is represented by the *Shape* object, then the object type is specified by the *msoShapeType* constants. Elements such as *msoChart*, *msoComment*, *msoEmbeddedOLEObject*, *msoFreeForm*, *msoGroup*, *msoOLEControlObject*, and *msoTextBox* prove that internally to Excel every object that is located outside of a cell is controlled by *Shape* objects.

Positioning: For each object is saved the upper left corner (*Left* and *Top*) as well as the width and height (*Width* and *Height*). These coordinates are figured from the upper left-hand corner of the form or worksheet. *TopLeftCell* and *BottomRightCell* specify the cells under the upper left-hand corner and lower right-hand corner. *Placement* determines how the control should behave when the worksheet is changed (*xlMoveAndSize*, *xlMove*, or *xlFreeFloating*).

Format: The possibilities for visual appearance are practically without bound. Each of the following properties leads to a particular object (whose name is given in parentheses if it is different from that of the property): *Adjustments*, *Callout* (*CalloutFormat*), *Fill* (*FillFormat*), *Hyperlink*, *Line* (*LineFormat*), *PictureFormat*, *Shadow* (*ShadowFormat*), *TextEffect* (*TextEffectFormat*), *TextFrame*, and *ThreeD* (*ThreeDFormat*). Perhaps this superfluity of objects is too much of a good thing.

Other: Depending on which objects are represented by *Shape*, there are further properties available: *ConnectorFormat* (if the object is bound to other objects), *ControlFormat* (for controls), *GroupItems* (for object groups), *Nodes* (for freehand objects), as well as *LinkFormat* and *OLEFormat* (for OLE objects).

POINTER Note that the *Shape* objects are defined in the Excel library, but the associated constants in the Office library. When old Excel 5/7 files are opened, the Office library is not activated under normal circumstances. This must be accomplished with `TOOLS\REFERENCES`.

Example

The drawing objects in Figure 10-8 were created with the loop in *btnShowAllAutoShapes_Click*. And now a word about the syntax of *AddShape*: The first parameter specifies the autoshape type (1 through 37), while the following four parameters determine the location (*Left/Top*) and size (*Width/Height*) of the object. The coordinate system begins in the upper left-hand corner of the worksheet.

```
' Shapes.xls, Sheet1
Private Sub btnShowAllAutoShapes_Click()
    Dim i&
    For i = 0 To 136
        ActiveSheet.Shapes.AddShape i + 1, _
            40 + 50 * (i Mod 12), 50 + 50 * (i \ 12), 40, 40
    Next
End Sub
```

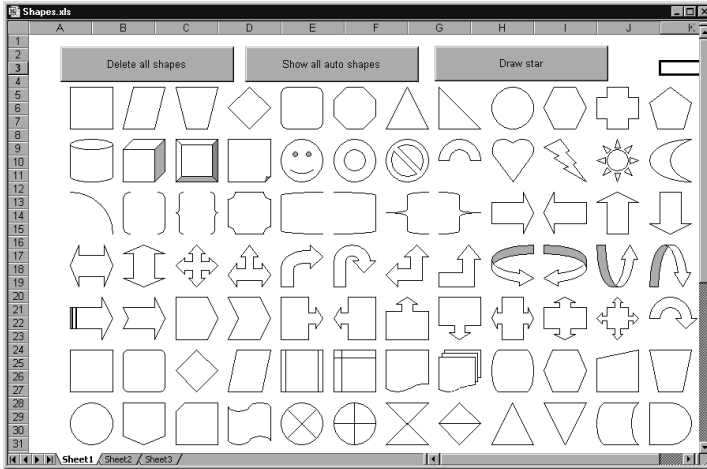


Figure 10-8. Some of the predefined autoshapes

To delete the drawing objects the following procedure can be used. The crucial step is the *Type* test: Without it the buttons in the worksheet would be deleted as well!

```
Private Sub btnDeleteShapes_Click()
    Dim s As Shape
    For Each s In ActiveSheet.Shapes
        If s.Type = msoAutoShape Or s.Type = msoLine Then s.Delete
    Next
End Sub
```

The procedure *btnStar_Click* draws a star made up of colored arrows (Figure 10-9). Note that arrows are not among the autoshapes, but form their own category of *Shape*. For this reason *AddLine* must be used instead of *AddShape*. *ForeColor* refers to a *ColorFormat* object, with which the color of an object can be set.

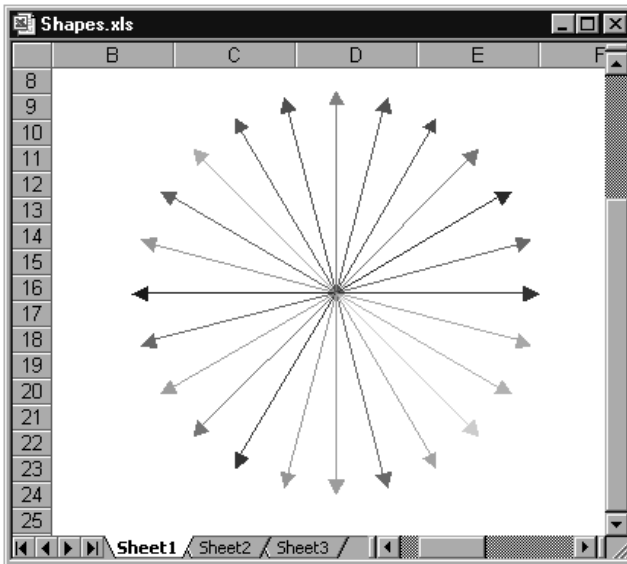


Figure 10-9. A star of colored arrows

POINTER *The program code may lead you to believe that Excel offers infinitely many colors for your use. Unfortunately, that is not the case. Rather, there is available a palette of only 56 colors (apparently a relic of earlier versions of Excel). Therefore, a reference to an RGB color means only that the closest matching color from this palette is used.*

```
Private Sub btnStar_Click()
    Dim degree#
    Dim s As Shape
    Const Pi = 3.1415927
    Randomize
    For degree = 0 To 2 * Pi Step Pi / 12
        Set s = ActiveSheet.Shapes.AddLine(200, 200, _
            200 + 100 * Sin(degree), 200 + 100 * Cos(degree))
        s.Line.EndArrowheadStyle = msoArrowheadTriangle
        s.Line.EndArrowheadLength = msoArrowheadLengthMedium
        s.Line.EndArrowheadWidth = msoArrowheadWidthMedium
        s.Line.ForeColor.RGB = RGB(Rnd * 255, Rnd * 255, Rnd * 255)
    Next
End Sub
```


10.6 Diagrams

Beginning with Excel 2002 you can use `INSERT|DIAGRAM` to insert an organization chart or one of five additional types of diagram into an Excel worksheet (cycle diagram, radial diagram, pyramid diagram, Venn diagram, and target diagram). These diagrams appear in a basic format and can be enhanced with your own text, formatting, and additional subobjects. In working with diagrams the `ORGANIZATION CHART` toolbars are helpful (only for organization charts), as well as the `DIAGRAM` toolbar (for the other five diagram types).

To create or edit a diagram in VBA code, you must use the new *DiagramXxx* objects, which are the focus of this section: ***Diagram*** describes an entire diagram, while ***DiagramNode*** refers to one of the diagram's elements. The enumerations ***DiagramNodes*** and ***DiagramNodeChildren*** help in the management of diagram elements.

TIP *Before you set out on the great adventure of programming diagrams, here are a few tips:*

- *Macro recording does not work either in creating or in editing a diagram. This makes the creation of Diagram objects a labor-intensive process.*
- *The Diagram objects themselves do not seem to be quite mature. As a particularly obvious example, diagrams that you create yourself cannot be given titles, and the title of an existing diagram cannot be changed.*
- *Save your project often! In the course of my experimentation I have suffered numerous crashes.*

Creating Diagrams

To create a new diagram, use the method ***AddDiagram*** of the *Shapes* object. You must specify the desired diagram type (*msoDiagramXxx* constant) as well as the size and location. As a result you receive a *Shape* object whose property *Diagram* refers to a like-named *Diagram* object.

```
Dim s As Shape
Dim d As Diagram
Dim ws As Worksheet
Set ws = Worksheets(1)
Set s = ws.Shapes.AddDiagram(msoDiagramRadial, 10, 10, 200, 100)
Set d = s.Diagram
```

Inserting Diagram Elements

A new diagram, regardless of type, is originally empty. The next step consists in filling the diagram with elements (with *DiagramNode* objects). It is annoying that the *Diagram* object refers, with the property *Nodes*, to a *DiagramNodes* enumeration, yet this enumeration does not, as is otherwise usual, have use of an *Add* method.

Finally, the example programs in the Help section show the only effective (yet completely illogical) way of proceeding: When you generate a new diagram with *AddDiagram*, you receive a *Shape* object (as described earlier). For this object one has the property *DiagramNode*, which refers to an object of this type. Apparently, together with each *Diagram* object an invisible and in some sense virtual *diagramNode* object is generated that serves as the starting point for the addition of additional elements. (But note that *DiagramNodes.Count* returns 0.)

How one proceeds next depends on the type of diagram: In the case of organization charts a root object must be created. All further objects are added as sub-objects (*Children*) of this root object. The following lines of code generate a radial diagram with a circle in the middle (*root*) and three associated circles around it (*child1* through *child3*).

```
' ws refers to a Worksheet object
' for msoDiagramRadial and msoDiagramOrgChart
Dim s As Shape
Dim root As DiagramNode, child1 As DiagramNode, _
    child2 As DiagramNode, child3 As DiagramNode
Set s = ws.Shapes.AddDiagram(msoDiagramRadial, 10, 10, 200, 100)
Set startnode = s.DiagramNode
Set root = startnode.Children.AddNode
Set child1 = root.Children.AddNode
Set child2 = root.Children.AddNode
Set child3 = root.Children.AddNode
```

With the other diagram types, however, all diagram objects are at the same level. The next example shows how a four-part pyramid diagram is generated:

```
' msoDiagramPyramid, msoDiagramCycle, msoDiagramTarget, msoDiagramVenn
Dim s As Shape
Dim child1 As DiagramNode, child2 As DiagramNode, _
    child3 As DiagramNode, child4 As DiagramNode
Set s = ws.Shapes.AddDiagram(msoDiagramPyramid, 10, 10, 200, 100)
Set startnode = s.DiagramNode
Set child1 = startnode.Children.AddNode
Set child2 = child1.AddNode
Set child3 = child1.AddNode
Set child4 = child1.AddNode
```

Providing Labels for Diagram Elements

The text of a diagram element is managed via a *TextFrame* object (see the previous section). Beginning with a *DiagramNode* object, the following list of properties results in the text property: *child1.TextShape.TextFrame.Characters.Text*.

The great problem is that a change in Text in Excel 2002 is not easily achievable. (Even the use of the alternative property *Caption* or the method *Insert* does not help.) That this error has not even been documented (let alone fixed) three-quarters of a year after the release of Excel 2002 leads one to believe that the new *DiagramXxx* objects have not caught the imagination of programmers.

Without the possibility of attaching labels to diagram elements, further programming makes no sense at all. One may hope that the many inconsistencies plaguing the *Diagram* objects will be corrected in future versions of Excel.

Deleting Diagrams

There is no *Delete* method for *Diagram* objects. Instead, the underlying *Shape* object must be deleted. With the property *HasShape* you can test whether the *Shape* object is being used to represent a diagram or for some other purpose. The following loop deletes the diagram in the first worksheet of a file:

```
Dim s As Shape
Dim ws As Worksheet
Set ws = Worksheets(1)
For Each s In ws.Shapes
    If s.HasDiagram Then s.Delete
Next
```