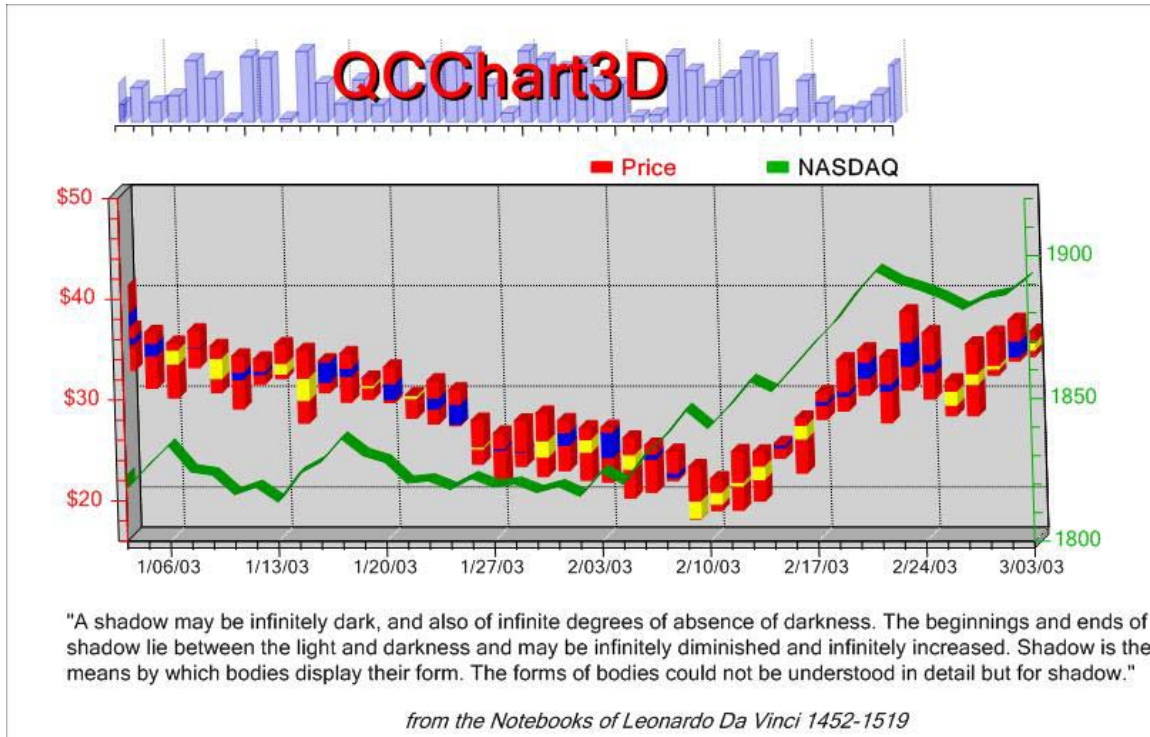


QCChart3D 3D Charting Tools for .Net



Contact Information

Company Web Site: <http://www.quinn-curtis.com>

Electronic mail

General Information: info@quinn-curtis.com

Sales: sales@quinn-curtis.com

Technical Support Forum

<http://www.quinn-curtis.com/ForumFrame.htm>

Revision Date 12/08/2006 Rev. 1.7

CChart3D for .Net Documentation and Software Copyright Quinn-Curtis, Inc. 2006

Quinn-Curtis, Inc. QCChart3D Tools for .Net END-USER LICENSE AGREEMENT

IMPORTANT-READ CAREFULLY: This Software End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Quinn-Curtis, Inc. for the Quinn-Curtis, Inc. SOFTWARE identified above, which includes the Quinn-Curtis, Inc. **QCChart3D 3D Charting Tools**. By installing, copying, or otherwise using the SOFTWARE, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use the SOFTWARE. If the SOFTWARE was mailed to you, return the media envelope, UNOPENED, along with the rest of the package to the location where you obtained it within 30 days from purchase.

1. The SOFTWARE is licensed, not sold.

2. GRANT OF LICENSE.

(A) **Developer License.** After you have purchased the license for SOFTWARE, and have received the file containing the licensed copy, you are licensed to copy the SOFTWARE only into the memory of the number of computers corresponding to the number of licenses purchased. The primary user of the computer on which each licensed copy of the SOFTWARE is installed may make a second copy for his or her exclusive use on a portable computer. Under no other circumstances may the SOFTWARE be operated at the same time on more than the number of computers for which you have paid a separate license fee. You may not duplicate the SOFTWARE in whole or in part, except that you may make one copy of the SOFTWARE for backup or archival purposes. You may terminate this license at any time by destroying the original and all copies of the SOFTWARE in whatever form.

(B) **30-Day Trial License.** You may download and use the SOFTWARE without charge on an evaluation basis for thirty (30) days from the day that you DOWNLOAD the trial version of the SOFTWARE. The termination date of the trial SOFTWARE is embedded in the downloaded SOFTWARE and cannot be changed. You must pay the license fee for a Developer License of the SOFTWARE to continue to use the SOFTWARE after the thirty (30) days. If you continue to use the SOFTWARE after the thirty (30) days without paying the license fee you will be using the SOFTWARE on an unlicensed basis.

Redistribution of 30-Day Trial Copy. Bear in mind that the 30-Day Trial version of the SOFTWARE becomes invalid 30-days after downloaded from our web site, or one of our sponsor's web sites. If you wish to redistribute the 30-day trial version of the SOFTWARE you should arrange to have it redistributed directly from our web site. If you are using SOFTWARE on an evaluation basis you may make copies of the evaluation SOFTWARE as you wish; give exact copies of the original evaluation SOFTWARE to anyone; and distribute the evaluation SOFTWARE in its unmodified form via electronic means (Internet, BBS's, Shareware distribution libraries, CD-ROMs, etc.). You may not charge any fee for the copy or use of the evaluation SOFTWARE itself. You must not represent in any way that you are selling the SOFTWARE itself. You must distribute a copy of this EULA with any copy of the SOFTWARE and anyone to whom you distribute the SOFTWARE is subject to this EULA.

(C) **Redistributable License.** The standard Developer License permits the programmer to deploy and/or distribute applications that use the Quinn-Curtis SOFTWARE, royalty free. We cannot allow developers to use this SOFTWARE to create a graphics toolkit (a library or any type of graphics component that will be used in combination with a program development environment) for resale to other developers.

If you utilize the SOFTWARE in an application program, or in a web site deployment, should we ask, you must supply Quinn-Curtis, Inc. with the name of the application program and/or the URL where the SOFTWARE is installed and being used.

3. RESTRICTIONS. You may not reverse engineer, de-compile, or disassemble the SOFTWARE, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation. You may not rent, lease, or lend the SOFTWARE. You may not use the SOFTWARE to perform any illegal purpose.

4. SUPPORT SERVICES. Quinn-Curtis, Inc. may provide you with support services related to the SOFTWARE. Use of Support Services is governed by the Quinn-Curtis, Inc. policies and programs described in the user manual, in online documentation, and/or other Quinn-Curtis, Inc.-provided materials, as they may be modified from time to time. Any supplemental SOFTWARE code provided to you as part of the Support Services shall be considered part of the SOFTWARE and subject to the terms and conditions of this EULA. With respect to technical information you provide to Quinn-Curtis, Inc. as part of the Support Services, Quinn-Curtis, Inc. may use such information for its business purposes, including for product support and development. Quinn-Curtis, Inc. will not utilize such technical information in a form that personally identifies you.

5. TERMINATION. Without prejudice to any other rights, Quinn-Curtis, Inc. may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE.

6. COPYRIGHT. The SOFTWARE is protected by United States copyright law and international treaty provisions. You acknowledge that no title to the intellectual property in the SOFTWARE is transferred to you. You further acknowledge that title and full ownership rights to the SOFTWARE will remain the exclusive property of Quinn-Curtis, Inc. and you will not acquire any rights to the SOFTWARE except as expressly set forth in this license. You agree that any copies of the SOFTWARE will contain the same proprietary notices which appear on and in the SOFTWARE.

7. EXPORT RESTRICTIONS. You agree that you will not export or re-export the SOFTWARE to any country, person, entity, or end user subject to U.S.A. export restrictions. Restricted countries currently include, but are not necessarily limited to Cuba, Iran, Iraq, Libya, North Korea, Sudan, and Syria. You warrant and represent that neither the U.S.A. Bureau of Export Administration nor any other federal agency has suspended, revoked or denied your export privileges.

8. NO WARRANTIES. Quinn-Curtis, Inc. expressly disclaims any warranty for the SOFTWARE. THE SOFTWARE AND ANY RELATED DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OR MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE REMAINS WITH YOU.

9. LIMITATION OF LIABILITY. IN NO EVENT SHALL QUINN-CURTIS, INC. OR ITS SUPPLIERS BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES OF ANY KIND ARISING OUT OF THE DELIVERY, PERFORMANCE, OR USE OF THE SUCH DAMAGES. IN ANY EVENT, QUINN-CURTIS'S LIABILITY FOR ANY CLAIM, WHETHER IN CONTRACT, TORT, OR ANY OTHER THEORY OF LIABILITY WILL NOT EXCEED THE GREATER OF U.S. \$1.00 OR LICENSE FEE PAID BY YOU.

10. U.S. GOVERNMENT RESTRICTED RIGHTS. The SOFTWARE is provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer SOFTWARE clause of DFARS 252.227-7013 or subparagraphs (c)(i) and (2) of the Commercial Computer SOFTWARE- Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is: Quinn-Curtis, Inc., 18 Hearsthstone Dr., Medfield MA 02052 USA.

11. MISCELLANEOUS. If you acquired the SOFTWARE in the United States, this EULA is governed by the laws of the state of Massachusetts. If you acquired the SOFTWARE outside of the United States, then local laws may apply.

Should you have any questions concerning this EULA, or if you desire to contact Quinn-Curtis, Inc. for any reason, please contact Quinn-Curtis, Inc. by mail at: Quinn-Curtis, Inc., 18 Hearthstone Dr., Medfield MA 02052 USA, or by telephone at: (508)359-6639, or by electronic mail at: support@Quinn-Curtis.com.

Table of Contents

Contact Information.....	i
1. Introduction.....	1
Tutorials.....	1
QCChart3D for .Net Background	1
QCChart3D for .Net Dependencies.....	2
System.Windows.Forms Namespace.....	2
System.Drawing Namespace	3
System.Drawing.Drawing2D Namespace	3
System.Drawing.Imaging Namespace	3
System.Drawing.Color Class.....	3
System.Drawing.Font Class.....	3
System.Drawing.Drawing2D.GraphicsPath Class.....	3
System.Drawing.Printing Namespace	3
System.Globalization Namespace.....	3
System.Drawing.Text Namespace	3
System.IO Namespace	3
System.Collections Namespace	3
Directory Structure.....	4
30-Day Trial Version	8
Developer Version	8
Chapter Summary	9
2. Class Architecture of the QCChart3D for .Net Class Library	11
Major Design Considerations	11
QCChart3D for .Net Class Summary.....	13
Chart Window Classes.....	14
Data Classes	14
Scale Classes.....	17
Coordinate Transform Classes.....	17
Attribute Class	18
Auto-Scaling Classes	19
Chart Object Classes.....	20
Axis Classes.....	22
Axis Label Classes.....	26
Chart Plot Classes	27
Group Plot Classes.....	29
Pie Chart Classes.....	41
Simple Plot Classes.....	42
Legend Classes.....	48
Grid Classes	49
Chart Text Classes	50
Miscellaneous Chart Classes.....	51
Mouse Interaction Classes	52
File and Printer Rendering Classes	54
Miscellaneous Utility Classes.....	54
Differences between the QCChart2D and QCChart3D API.....	57

Class Hierarchy of QCChart3D	61
3. Chart Datasets	63
2½ D Datasets and 3D Datasets	63
Simple Numeric Dataset	64
Class SimpleDataset	64
Class SimpleDataset3D	69
Simple Date/Time Dataset	72
Class TimeSimpleDataset	72
Class TimeSimpleDataset3D	77
ContourDataset Dataset	81
Class ContourDataset	81
Numeric Group Dataset	87
Class GroupDataset	87
Class GroupDataset3D	90
Date/Time Group Dataset	94
Class TimeGroupDataset	94
Class TimeGroupDataset3D	99
4. Scaling and Coordinate Systems	105
Plot area volume and the graph area volume	105
Coordinate Systems	107
User Coordinates	107
World Coordinates	107
Working Coordinates	107
Physical Coordinates	107
Normalized coordinates	108
Chart Scaling and the .Net Drawing2D.Matrix	108
Important numeric considerations	109
Value limiting	109
Bad value checking	109
Taking the logarithm of 0 or a negative number	109
Positioning the Plot Area Volume in Graph Area Volume	109
Rotating the Chart in the Graph Area Volume	115
AbsRotateCoordinateSystem	115
RelRotateCoordinateSystem	116
RelRotateCoordinateSystem	116
AbsTranslateCoordinateSystem	116
Linear and Logarithmic Coordinate Scaling	117
Class CartesianCoordinates	117
Linear Coordinate Scaling	117
Logarithmic Coordinate Scaling	123
Coordinate Systems using times and dates	125
The ChartCalendar Class	125
5 vs. 7 Day Work Weeks	126
Non-24 Hour Days	126
Non-Uniformity of Date/Time Tick Marks	127
Class TimeCoordinates	128

Table of Contents

Miscellaneous Coordinate System Topics	140
Inverted Coordinate Systems	140
5. The Chart View	143
Setting the Relative Depth of the ChartView	145
SetFractionalZViewportDepth method	145
Rendering Order of GraphObj Objects	149
Dynamic or Real-Time Updates of Chart Objects	150
Placing Multiple Charts in a ChartView	151
Multiple Coordinate Systems in the Same Chart	152
ChartView Object Resize Modes	153
ChartView View Modes	155
Finding Chart Objects	155
6. Backgrounds, Walls, Borders and Bitmaps	157
Class Background	157
Class Wall3D	163
7. Axes	167
Chart Axes	168
Linear Axes	169
Class LinearAxis	169
Logarithmic Axes	175
Class LogAxis	175
Date/Time Axes	182
Class TimeAxis	182
8. Axis Labels	193
Axis Labels	193
Label Formats	193
Class AxisLabels	194
Numeric Axis Labels	195
Class NumericAxisLabels	195
Time and Date Axis Labels	200
Class TimeAxisLabels	200
String Axis Labels	206
Class StringAxisLabels	206
9. Axis Grids	211
Linear, Logarithmic and Time Axis Grids	211
Class Grid	211
10. Simple Plot Objects	219
Simple Line Plots	219
Class SimpleLinePlot	219
Simple Ribbon Plots	225
Class SimpleRibbonPlot	225
Simple Pipeline Plots	228
Class SimpleLinePlot	228
Simple Bar Plots	231
Class SimpleBarPlot	231
Simple Scatter Plots	236

Class SimpleScatterPlot.....	236
Simple Line Marker Plots	241
Class SimpleLineMarkerPlot.....	241
11. Group Plot Objects.....	247
Arrow Plots	248
Class ArrowPlot.....	248
Bubble Plots	252
Class BubblePlot.....	252
Candlestick Plots.....	255
Class CandlestickPlot.....	255
Cell Plots	259
Class CellPlot.....	259
Error Bar Plots	263
Class ErrorBarPlot	263
Floating Bar Plots	264
Class FloatingBarPlot	264
Class FloatingStackedBarPlot.....	272
Group Bar Plots.....	275
Class GroupBarPlot	275
Histogram Plots.....	280
Class HistogramPlot.....	280
Line Gap Plots.....	285
Class LineGapPlot	285
Multi-Line Plots	289
Class MultiLinePlot	289
Class MultiRibbonPlot.....	293
Open-High-Low-Close Plots	295
Class OHLCPlot.....	295
Class OHLCBarPlot.....	298
Stacked Bar Plots	301
Class StackedBarPlot.....	301
Stacked Line Plots.....	304
Class StackedLinePlot	304
Class StackedRibbonPlot.....	308
12. Contour Plotting.....	313
Line and Filled Contour Plots	313
Class ContourPlot	313
13. Data Markers and Data Cursors.....	321
Data Markers.....	321
Class Marker	321
Data Cursors.....	324
Class DataCursor	324
14. Moving Chart Objects and Data Points	329
Moving Chart Objects.....	329
Class MoveObj.....	329
Moving Simple Plot Object Data Points	332

Table of Contents

Class MoveData	332
15. Zooming.....	337
Simple Zooming of a single physical coordinate system.....	337
Class ChartZoom	337
Super Zooming of multiple physical coordinate systems	341
Limiting the Zoom Range.....	346
16. Data Tooltips.....	349
Simple Data Tooltips	349
Class DataToolTip	349
Custom Tooltip displays	355
17. Pie Charts.....	361
Using the Pie Chart Class	361
Class PieChart.....	361
19. Legends.....	371
Standard Legends.....	371
Class StandardLegend.....	371
Bubble Plot Legends.....	376
Class BubblePlotLegend.....	376
20. Text Classes	381
Important note about ChartText rotations.....	381
Simple Text Classes.....	382
Class ChartText.....	382
Chart Title Classes	387
Class ChartTitle	387
Class AxisTitle.....	390
Numeric, Time and String Label Classes	392
Class ChartLabel.....	392
21. Adding Lines, Shapes, Images and Arrows to a Chart	401
Important note about rotations.....	401
Generic Shape Class	402
Class ChartShape	402
Chart Image Class.....	405
Class ChartImage.....	405
Generic Arrow Class.....	408
Class Arrow	408
22. File and Printer Rendering Classes	413
Printing a Chart.....	413
Class ChartPrint	413
Using All of the Paper When Printing.....	415
Capturing the Chart as a Buffered Image	421
Class BufferedImage.....	421
23. Using QCChart3D for .Net to Create Windows Applications	425
(** Critical Note **) Running the Example Programs.....	425
Visual Basic for .Net.....	425
Visual C# for .Net.....	427
24. Using QCChart3D for .Net to Create Web Applications.....	431

Do Not Try and Compile our ASP.Net Applications	431
On Demand Creation of QCChart3D Images from a Server	432
Visual C# for .Net	432
Visual Basic for .Net	440
Embedding a QCChart3D Object into Internet Explorer.....	447
Visual C# - Create a Windows Control Library and User Control.....	448
Visual Basic - Create a Windows Control Library and User Control.....	453
Create the HTML File That Displays the User Control.....	457
25. Frequently Asked Questions	461
FAQs	461
INDEX	489

The QCChart3D for .Net Charting Library

1. Introduction

Tutorials

Tutorials that describe how to get started with the QCChart3D for .Net charting software are found in Chapter 23 (*Using QCChart3D for .Net to Create Windows Applications*) and Chapter 24 (*Using QCChart3D for .Net to Create Web Applications*).

QCChart3D for .Net Background

Charting is the display of quantitative information using a combination of lines, points, bars, colors, curves, text and images. Charts are a relatively new invention, first organized, categorized, and extensively used by William Playfair (1759-1823) in the late 18th century. The purpose of charts has remained unchanged since that time. A chart replaces a tabular numeric display of information with a graphical image that visually shortcuts to the underlying pattern in the data.

A chart does not have to be an output only display medium. A chart can be interactive where a user interacts with a chart using the mouse and keyboard: changing colors, changing axes limits, highlighting data values are just a few of the interactive options programmers want in the charts they create.

The goal of the Microsoft .Net framework is to create an integrated platform for the creation, distribution and execution of application programs. The applications can be standalone programs suitable for a single computer and a single user, networked client server applications, or Internet applications involving local and remote servers. Fast, powerful, and flexible charting graphics is common requirement for application programs in all market segments.

A long term goal of Microsoft .Net is platform independence. This implies that application programs written in C# or VB will run not only on Windows platforms, but UNIX and Linux based machines. Since Microsoft .Net is optimized for Windows, making use of large portions of the Win32 operating system, porting it to UNIX based machines is a lot of work. Microsoft's goal of platform independence may be achieved not by supporting UNIX, but by eliminating the competition until all computers run some form of Windows.

The major competitor to Microsoft .Net is Java, the language invented by Sun Computer. In some respects, Java already is what Microsoft .Net wants to be. You can write

2 Introduction

application programs using Java that are very platform independent. This is because Java interpreters have been written for most modern operating systems. The disadvantage of Java is that it creates application programs much slower than C/C++, C# and VB. Microsoft was once a Java supporter and a member of the Java language consortium. Microsoft wanted to create their own version of Java optimized for Windows that would not run on other platforms. Sun disapproved of any company creating “flavors” of Java that relied on platform specific features and disbanded the consortium rather than let Microsoft, or any other company, proceed down that path. At that point Microsoft decided to compete directly with Java using the .Net languages. Microsoft is loath to aid the marketing of a competitor and steadfastly refuses to distribute Sun’s version of the Java virtual machine with the Windows XP operating system. In order to run Java applications on a Windows platform you need to download the Java virtual machine from Sun and install it on your computer. While this is easy to do, it does not represent the far wider distribution of the Java virtual machine that Sun is looking for, and the company has had Microsoft in court seeking some remedy for many years.

Microsoft .Net includes a very rich API for writing applications that make use of GUI’s, data structures, databases, files and streams, networking and web services. The graphics part of the API is a combination of the features found in the Win32 graphics API and those in the Java API. Most of the programming features found in the Java Graphics2D API are duplicated in one form or another in .Net API. While the Java graphics classes are more extensive and in many cases more elegant than those in .Net, Microsoft did a good job of providing a core set of classes that allow most Java applications to be ported to .Net.

Features found in the Microsoft .Net API and used in the **QCChart3D for .Net** library include the following:

- Arbitrary line thickness and line styles for lines.
- Gradients, fill patterns and color transparency for solid objects.
- Generalized geometry support used to create arbitrary shapes
- Printer and image output support
- Improved font support for a large number of fonts, using a variety of font styles, size and rotation attributes.
- Imaging support for a large number of image formats
- Advanced matrix support for handling 2D transformations.

QCChart3D for .Net Dependencies

The **QCChart3D for .Net** class library is self-contained. It uses only standard classes that ship with the Microsoft .Net API. The software uses the major .Net namespaces listed below.

System.Windows.Forms Namespace

The **System.Windows.Forms** namespace contains classes for creating .Net Forms, Controls and Dialog boxes.

System.Drawing Namespace

The **System.Drawing** namespace provides access to GDI+ basic graphics functionality. More advanced functionality is provided in the **System.Drawing.Drawing2D**, **System.Drawing.Imaging**, and **System.Drawing.Text** namespaces.

System.Drawing.Drawing2D Namespace

The **System.Drawing.Drawing2D** namespace provide advanced 2-dimensional and vector graphics functionality. This namespace includes the gradient brushes, the **Matrix** class (used to define geometric transforms), and the **GraphicsPath** class.

System.Drawing.Imaging Namespace

The **System.Drawing.Imaging** namespace provides advanced GDI+ imaging functionality. Basic graphics functionality is provided by the **System.Drawing** namespace.

System.Drawing.Color Class

Provides a class to define colors in terms of their individual ARGB (Alpha, Red, Green, Blue) components.

System.Drawing.Font Class

Defines a particular format for text, including font face, size, and style attributes.

System.Drawing.Drawing2D.GraphicsPath Class

This class is used to define a series of connected lines and curves.

System.Drawing.Printing Namespace

This namespace provides classes and interfaces for a general printing API.

System.Globalization Namespace

The **System.Globalization** namespace contains classes that define culture-related information, including the language, the country/region, the calendars in use, the format patterns for dates, currency, and numbers, and the sort order for strings.

System.Drawing.Text Namespace

The **System.Drawing.Text** namespace provides advanced GDI+ typography functionality. Basic graphics functionality is provided by the **System.Drawing** namespace.

System.IO Namespace

The **IO** namespace contains types that allow synchronous and asynchronous reading and writing on data streams and files.

System.Collections Namespace

4 Introduction

The **System.Collections** namespace contains interfaces and classes that define various collections of objects, such as lists, queues, bit arrays, hashtables and dictionaries.

Directory Structure

The QCChart3D for *.Net* class library uses the following directory structure:

Drive:

Quinn-Curtis\ - Root directory

DotNet\ - Quinn-Curtis .Net based products directory

Docs\ - Quinn-Curtis .Net related documentation directory

Lib\ - Quinn-Curtis .Net related compiled libraries and components directory

QCChart3D\ - Language specific code directory

Visual CSharp\ - C# specific directory

QCChart3DClassLib\ - contains the source code to the
QCChart3DNet.dll library (installed only if the
source code has been purchased)

Examples\ - C# examples directory

Bargraphs\ - Horizontal, vertical, stacked, group, histogram,
floating

CalendarData\ - Line plots, bar plots and log plots that use
time/date data. Also reading time/date data from a
file.

ChartAxes\ - Linear, logarithmic, time/date, custom hours and
time/date. Also axes labels.

ContourPlots\ - Line and filled contour plots.

CustomDataToolTips\ - Creating custom data tooltips for an
OHLC plot, multiple stacked graphs and a pie chart.

DataCursorsAndMarkers\ - Using data cursors and markers

DynamicCharts\ - Scrolling lines, bars, scatter plots, data logging, instrument simulation, chart animation.

EditChartExample\ - Dialog box for chart example

FinancialExamples\ - OHLC plots, candlestick plots, financial log plots, option chart, technical analysis chart.

FormControlExamples\ - Adding check boxes, scrollbars and tables to charts.

ImageCharts\ - Using images as chart data elements, chart backgrounds and annotations.

LabeldDatapoints\ - Label the data points of a simple line chart.

LinePlotSalesVolume\ - Simple line plot example with printing and save image menu.

LogPlots\ -Logarithmic plots for financial charts and engineering charts.

MiscCharts\ - A line gap chart.

MouseListeners\ - Data tooltips, data cursors, moving data points, moving chart objects.

MultiLinePlots\ - Group multi-line plots, stacked line plots, multiple single line graphs.

MultipleAxes\ - Multiple axes graphs

PieCharts\ - Simple pie charts and pie charts combined with line and bar plots.

ResizeExamples\ - Fixed size frame, resizable frame with fixed sized objects, resizable frame with resizable objects, scrollable panel as a view into a much larger fixed size frame.

ScatterPlots\ - Simple scatter, line and line-marker plots, scatter plots with variable size and color symbols, cell plots, arrow plots. Also, labeling the data point values or a line marker plot.

SimpleLinePlots\ - Simple line plots with linear and time/date coordinate systems. Also filled and step lines.

6 Introduction

UserChartExample1\ - Simple example displays line and scatter plots

WebsiteStats\ - The example displays summary web site information in four separate windows.

WindowsControlLibrary1\ - The example embeds a ChartView derived chart in a web page, for viewing with Internet Explorer.

ZoomExamples\ - Zooming simple linear axes, super zooming of multiple axes, zooming of time/data based data.

ASP.Net\ - Holds ASP.Net examples specific directory

Chart3DAspWebApp\ - ASP.Net Web application

Chart3DHTMLAspApp - ASP.Net App in HTML page

Chart3DWebApplication1 - Tutorial example program for ASP.Net web application

Visual Basic\ - VB specific code

Examples\ - VB examples

Bargraphs\ - Horizontal, vertical, stacked, group, histogram, floating

CalendarData\ - Line plots, bar plots and log plots that use time/date data. Also reading time/date data from a file.

ChartAxes\ - Linear, logarithmic, time/date, custom hours and time/date. Also axes labels.

ContourPlots\ - Line and filled contour plots.

CustomDataToolTips\ - Creating custom data tooltips for an OHLC plot, multiple stacked graphs and a pie chart.

DataCursorsAndMarkers\ - Using data cursors and markers

DynamicCharts\ - Scrolling lines, bars, scatter plots, data logging, instrument simulation, chart animation.

EditChartExample\ - Dialog box for chart example

FinancialExamples\ - OHLC plots, candlestick plots, financial log plots, option chart, technical analysis chart.

FormControlExamples\ - Adding check boxes, scrollbars and tables to charts.

ImageCharts\ - Using images as chart data elements, chart backgrounds and annotations.

LabeldDatapoints\ - Label the data points of a simple line chart.

LinePlotSalesVolume\ - Simple line plot example with printing and save image menu.

LogPlots\ -Logarithmic plots for financial charts and engineering charts.

MiscCharts\ - A line gap chart.

MouseListeners\ - Data tooltips, data cursors, moving data points, moving chart objects.

MultiLinePlots\ - Group multi-line plots, stacked line plots, multiple single line graphs.

MultipleAxes\ - Multiple axes graphs

PieCharts\ - Simple pie charts and pie charts combined with line and bar plots.

ResizeExamples\ - Fixed size frame, resizable frame with fixed sized objects, resizable frame with resizable objects, scrollable panel as a view into a much larger fixed size frame.

ScatterPlots\ - Simple scatter, line and line-marker plots, scatter plots with variable size and color symbols, cell plots, arrow plots. Also, labeling the data point values or a line marker plot.

SimpleLinePlots\ - Simple line plots with linear and time/date coordinate systems. Also filled and step lines.

UserChartExample1\ - Simple example displays line and scatter plots

WebsiteStats\ - The example displays summary web site information in four separate windows.

WindowsControlLibrary1\ - The example embeds a ChartView derived chart in a web page, for viewing with Internet Explorer.

ZoomExamples\ - Zooming simple linear axes, super zooming of multiple axes, zooming of time/data based data.

ASP.Net\ - Holds ASP.Net examples specific directory

VBChart3DAspWebApp\ - ASP.Net Web application

VBChart3DHTMLAspApp – ASP.Net App in HTML page

VBChart3DWebApplication1 – Tutorial example program for ASP.Net web application

There are two versions of the **QCChart3D** class library: the 30-day trial versions, and the developer version. Each version has different characteristics that are summarized below:

30-Day Trial Version

The trial version of QCChart3D for *.Net* is downloaded as a zip file named Trial_QCChart3DR10x.zip. Once you download the zip file, you un-zip it to a local hard drive and run the Setup.exe program from the resulting QCChart3DInstall folder. The 30-day trial version stops working 30 days after you run the Setup.exe program for the first time. The trial version includes a trial version message in the upper right corner of the graph window that cannot be removed.

Developer Version

The developer version of QCChart3D for *.Net* is downloaded as a zip file, name something similar to NETCH3DDEV1R1x0x353x1.zip. Once you download the zip file, you un-zip it to a local hard drive and run the Setup.exe program from the resulting QCChart3DInstall folder. You can use your original download links to download free updates for a period of 2-years.

Chapter Summary

The remaining chapters of this book discuss the **QCChart3D for .Net** interactive charting package designed to run on any hardware that has .Net or higher interpreter, available for it.

Chapter 2 presents the overall class architecture of the **QCChart3D for .Net** and summarizes all of the classes found in the software.

Chapter 3 describes the dataset classes that hold chart data.

Chapter 4 describes the various classes that implement the Cartesian and time coordinate systems supported by the software.

Chapter 5 describes the **ChartView** container class that manages the chart objects.

Chapter 6 describes a class that will create simple one color, gradient and texture backgrounds for a chart.

Chapters 7, 8 and 9 describe the classes that create chart axes, axis labels and axis grids.

Chapter 10 describes the classes used to display simple xy data (one y-value for each x-value) as line plots, bar plots and scatter plots.

Chapter 11 describes the classes used to display group data (one or more y-values for each x-value) as line plots, group bar plots, stacked bar plots, scatter plots, open-high-low-close plots, and candlestick plots.

Chapter 12 describes plotting surface data using line and filled contours.

Chapter 13, 14, 15 and 16 describe classes that add interactive elements to a chart. Chapter 13 describes data marker and data cursor classes used to mark and highlight data points using the mouse. Chapter 14 describes classes that can move chart objects and individual data points. Chapter 15 adds a “zooming” class where mouse events define a new scaling range for a chart, redrawing the chart axes and data automatically. Chapter 16 describes a generalized data tool-tip class that can display the x and/or y data values for a data point, or custom text associated with the data point.

Chapter 17 describes classes for the display of pie charts.

Chapter 19 describes classes for the display chart legends, used to create visual aids for the interpretation of different elements making up the chart.

Chapter 20 describes generalized classes for displaying formatted text in a chart.

Chapter 21 describes how to use a generalized shape class for the display of arbitrary lines, shapes, images and arrows.

Chapter 22 describes chart printing and the creation of JPEG files.

10 Introduction

Chapter 23 is a tutorial that describes how to use **QCChart3D** to create Windows applications using Visual Studio .Net, Visual C# and Visual Basic.

Chapter 24 is a tutorial that describes how to use **QCChart3D** to create ASP.NET web applications using Visual Studio .Net, Visual C# and Visual Basic. It also describes how to embed a ChartView derived user control into a web page that is view using Internet Explorer.

Chapter 25 is a collection of Frequently Asked Questions about **QCChart3D for .Net**.

2. Class Architecture of the QCChart3D for .Net Class Library

Major Design Considerations

This chapter presents an overview of the **QCChart3D for .Net** class architecture. It discusses the major design considerations of the architecture:

- **QCChart3D** uses the .Net **System.Drawing** API model and the **System.Windows.Forms** classes.
- New charting objects can be added to the library without modifying the source of the base classes.
- There are no limits regarding the number of data points in a plot, the number of plots in graph, the number of axes in a graph, the number of coordinate systems in a graph.
- There are no limits regarding the number of legends, arbitrary text annotations, bitmap images, geometric shapes, titles, data markers, cursors and grids in a graph.
- Users can interact with charts using classes using **System.EventHandler** delegate event driven model.
- The .Net Framework 2.0 that is included with Visual Studio 2005 does not include a 3D plotting library, analogous to the **System.Drawing.Drawing2D** drawing library. While there are high quality and high-speed 3D libraries for .Net, most notable **Direct3D** (**DirectX**) and **OpenGL**, they have significant shortcomings, most notable in the area of printing. This software implements its own 3D graphics engine, and then renders the resulting points, lines, text and polygons using the the **System.Drawing.Drawing2D** graphics library. This allows the high speed, high resolution using the .Net printer drivers. A tradeoff is the software does not use a raster z-buffer, the most efficient technique for implementing hidden line and surface removal. The hidden surface removable capability is limited to a simple painter's algorithm, where whole objects are sorted by their z-axis positioning value and drawn in the right order. That technique is not as universal as the hidden-surface removal found in **OpenGL** and **Direct3D**, and cannot handle intersecting solids.

The chapter also summarizes the classes in the **QCChart3D for .Net** library.

There are five primary features of the overall architecture of the **QCChart3D for .Net** classes. These features address major shortcomings in existing charting software for use with both .Net and other computer languages.

- First, **QCChart3D** uses the standard .Net window architecture. Charts are placed in a **ChartView** window that derives from the **System.Windows.Forms.UserControl** class. Position one or more **ChartView** objects in .Net container windows using the standard container layout managers. Mix charts with other components in the same container. Charts use the standard .Net event-processing model for handling mouse and keyboard events.
- Second, the library is extensible. Hundreds of different vertical markets use computer charting. The charts used in each market have a unique look and feel. A well-designed object oriented charting package allows the programmer to extend the software without modifying the source of the underlying classes. Instead, the programmer extends the software by deriving a new class from an existing base class. The new, derived class localizes custom source code and the source of the underlying classes remains unchanged. In the **QCChart3D** classes a user can subclass an existing class and create new, custom charting objects. Examples of custom charting objects are specialized plots that extend the **SimplePlot**, or **GroupPlot** classes to include new plot types for applications such as stock market technical analysis, statistical process control, and medical instrumentation, to name a few.
- Third, the library has no limits regarding the number of data points in a plot, the number of plots in graph, the number of axes in a graph, and the number of coordinate systems in a graph. A major weakness in many commercial graphics packages is that they have hard coded limits that restrict the number of data points, axes, or coordinate systems. A simple business bar chart may only contain 3 or 4 data points, depicting a sales forecast. An audio mixer application may require 32 million plotted points, represented by 32 traces of 1 million points each, each trace representing 20 seconds of audio sampled at 50 kHz.

The most effective way to compare data is to overlay it in the same graph. Often the data series have different dynamic ranges. If the data series are plotting using the same scale, it is difficult to see the correlation, or lack of, in the data. A better solution is to create a unique scale for each data series, with associated axes, and plot each data series with respect to its own scale. All of the plots will overlay the same area of the graph. Many advanced charting packages do not support more than one coordinate system per graph, while most others have some fixed limit such as 2, 4 or 8 scales per graph. The number of coordinate systems for a graph can be as large as the number of data series plotted in the graph. Charts can have hundreds of data series at once; therefore, a flexible charting package needs to allow for an equal number of simultaneous coordinate systems.

- Fourth, a well-constructed chart often displays more than just data. Other common chart objects include legends, arbitrary text annotations, bitmap images, geometric shapes, titles, data markers, cursors and grids. A chart can contain zero or more of these objects. It may contain 100 of one type, 5 of another type, and 1 of a third. **QCChart3D for .Net** contains no limits restricting the number of instances of a given chart object in a graph, and no limit on the total number of chart objects in a graph.

- Fifth, an end user needs to interact with the graph using the mouse and/or keyboard. The **QCChart3D** architecture includes classes that implement the .Net **System.EventHandler** delegate event driven model. A user can use the mouse to select data points, text annotations, axes, image objects, and other shapes, and position them in the graph. Create data markers and move them around the chart under mouse or program control. Automatically rescale one or more chart axes using mouse controlled zooming.

QCChart3D for .Net Class Summary

The following categories of classes realize these design considerations.

Chart view class	The chart view class is a UserControl subclass that manages the graph objects placed in the graph
Data classes	There are data classes for simple xy, xyz, and group data types. There are also data classes that handle System.DateTime date/time data.
Scale transform classes	The scale transform classes handle the conversion of physical coordinate values to working coordinate values for a single dimension.
Coordinate transform classes	The coordinate transform classes handle the conversion of physical coordinate values to working coordinate values for a parametric (2D) coordinate system.
Attribute class	The attribute class encapsulates the most common attributes (line color, fill color, line style, line thickness, etc.) for a chart object.
Auto-Scale classes	The coordinate transform classes use the auto-scale classes to establish the minimum and maximum values used to scale a 3D coordinate system. The axis classes also use the auto-scale classes to establish proper tick mark spacing values.
Charting object classes	The chart object classes includes all objects placeable in a chart. That includes axes, axes labels, plot objects (line plots, bar graphs, scatter plots, etc.), grids, titles, backgrounds, images and arbitrary shapes.
Mouse interaction classes	These classes, directly and indirectly System.EventHandler delegates that trap mouse events and permit the user to create and move data cursors,

move plot objects, display tooltips and select data points in all types of graphs.

File and printer rendering These classes render one or more chart images to a printer, to a variety of file formats including JPEG, and BMP, or to a .Net **Image** object.

Miscellaneous utility classes Other classes use these for data storage, file I/O, and data processing.

A summary of each category appears in the following section.

Chart Window Classes

System.Windows.Forms.UserControl
ChartView

The starting point of a chart is the **ChartView** class. The **ChartView** class derives from the .Net **System.Windows.Forms.UserControl** class, where the **UserControl** class is the base class for the .Net collection of standard components such as menus, buttons, check boxes, etc. The **ChartView** class manages a collection of chart objects in a chart and automatically updates the chart objects when the underlying window processes a paint event. Since the **ChartView** class is a subclass of the **UserControl** class, it acts as a container for other .Net components too.

Data Classes

ChartDataset
 SimpleDataset
 SimpleDataset3D
 ContourDataset
 TimeSimpleDataset
 TimeSimpleDataset3D
 GroupDataset
 GroupDataset3D
 TimeGroupDataset
 TimeGroupDataset3D

The dataset classes organize the numeric data associated with a plotting object. There are two major types of data supported by the **ChartDataset** class. The first major type is collection of simple xy (and/or z) data values, where for every x-value there is a unique y-value and, either a fixed z-value, as in the case of the **SimpleDataset** and **TimeSimpleDataset** classes, or a unique z-value, as in the case of the **SimpleDataset3D** and **TimeSimpleDataset3D** classes. The second major type is group data, where every x-value can have one or more y-values, and a fixed, or unique z-value.

ChartDataset The abstract base class for the other dataset classes. It contains data common to all of the dataset classes, such as the x-value array, the number of x-values, the dataset name and the dataset type.

SimpleDataset Represents simple xy data, where for every x-value there is one y-value. The dataset uses a single z-value, set using the property **ImplicitZValue** that applies to all xy data pairs in the dataset. It also includes a property, **ImplicitDepthValue**, that is used to set the depth of 3D data objects, such as 3D bars and ribbons. This type of dataset is often referred to as 2½ D, since it is used to place 2D data in a 3D coordinate system.

TimeSimpleDataset A subclass of **SimpleDataset**, it uses **ChartCalendar** dates (a wrapper around the System.DateTime value class) as the x-values, and floating point numbers as the y-values. Like the **SimpleDataset** class, it uses a single z-value, set using the property **ImplicitZValue** that applies to all xy data pairs in the dataset. . It also includes a property, **ImplicitDepthValue**, that is used to set the depth of 3D data objects, such as 3D bars and ribbons.

SimpleDataset3D A subclass of **SimpleDataset**, it adds an array of z-values to the x- and y- values of the simple dataset. Unlike the **SimpleDataset** and **TimeSimpleDataset**, the **SimpleDataset3D** class represents true 3D data points, where every data point is characterized by a unique x, y and z value. . It also includes a property, **ImplicitDepthValue**, that is used to set the depth of 3D data objects, such as 3D bars and ribbons.

TimeSimpleDataset3D A subclass of **TimeSimpleDataset**, it adds an array of z-values to the x- and y-values of the **TimeSimpleDataset**. The class represents true 3D data points, where every data point is characterized by a unique x, y and z value. It also includes a property, **ImplicitDepthValue**, that is used to

set the depth of 3D data objects, such as 3D bars and ribbons.

ContourDataset

A subclass of **SimpleDataset3D**, it adds a polysurface structure that organizes a grid of 3D data points as a surface. This class is used exclusively with the **ContourPlot** class.

GroupDataset

Represents group data, where every x-value can have one or more y-values. The dataset uses a single z-value, set using the property **ImplicitZValue** that applies to all xy data groups in the dataset. This type of dataset is often referred to as 2½ D, since it is used to place 2D data in a 3D coordinate system. It also includes a property, **ImplicitDepthValue**, that is used to set the depth of 3D data objects, such as 3D bars and ribbons.

GroupDataset3D

A subclass of **GroupDataset**, it adds an array of z-values to the x- and y- values of the group dataset. The class represents true 3D data points, where every data point is characterized by a unique x, y and z value. . It also includes a property, **ImplicitDepthValue**, that is used to set the depth of 3D data objects, such as 3D bars and ribbons.

TimeGroupDataset

A subclass of **GroupDataset**, it uses **ChartCalendar** dates (a wrapper around the System.DateTime value class) as the x-values, and floating point numbers as the y-values. Like the **SimpleDataset** class, it uses a single z-value, set using the property **ImplicitZValue** that applies to all xy data pairs in the dataset. It also includes a property, **ImplicitDepthValue**, that is used to set the depth of 3D data objects, such as 3D bars and ribbons.

TimeGroupDataset3D

A subclass of **TimeGroupDataset**, it adds an array of z-values to the x- and y- values of the group dataset. The class represents true 3D data points, where every data point is characterized by a unique x, y and z value. . It also includes a property, **ImplicitDepthValue**, that is used to set the depth of 3D data objects, such as 3D bars and ribbons.

Scale Classes

ChartScale

LinearScale

LogScale

TimeScale

The **ChartScale** abstract base class defines coordinate transformation functions for a single dimension. It is useful to be able to mix and match different scale transform functions for x-, y-, and z-dimensions of the **PhysicalCoordinates** class. The job of a **ChartScale** derived object is to convert a dimension from the current *physical* coordinate system into the current *working* coordinate system.

LinearScale	A concrete implementation of the ChartScale class. It converts a linear physical coordinate system into the working coordinate system.
LogScale	A concrete implementation of the ChartScale class. It converts a logarithmic physical coordinate system into the working coordinate system.
TimeScale	A concrete implementation of the ChartScale class. converts a date/time physical coordinate system into the working coordinate system.

Coordinate Transform Classes

Transform3D

UserCoordinates

WorldCoordinates

WorkingCoordinates

PhysicalCoordinates

CartesianCoordinates

TimeCoordinates

The coordinate transform classes maintain a 3D coordinate system. Different coordinate systems are used to position and draw objects in a graph. Examples of some of the coordinate systems include the device coordinates of the current window, normalized coordinates for the current window and plotting area, and scaled physical coordinates of the plotting area.

Transform3D	This class maintains a (4x4) homogeneous 3D transformation matrix that rotates, scales and translates coordinates from 3D device coordinates into 2D screen coordinates.
UserCoordinates	This class manages the interface to the System.Drawing classes and contains routines for drawing lines, rectangles and text using .Net device coordinates. It also contains an instance of a Transform3D object that transforms from 3D device coordinates to 2D screen coordinates.
WorldCoordinates	This class derives from the UserCoordinates class and maps a device independent world coordinate system on top of the .Net device coordinate system.
WorkingCoordinates	This class derives from the WorldCoordinates class and extends the physical coordinate system of the <i>plot area volume</i> (the area typically bounded by the charts axes) to include the complete <i>graph area volume</i> (the area of the chart outside of the <i>plot area volume</i>).
PhysicalCoordinates	This class is an abstract base class derived from WorkingCoordinates and defines the routines needed to map the physical coordinate system of a plot area volume into a working coordinate system. Different scale objects (ChartScale derived) are installed for converting physical x-, y-, and z-coordinate values into working coordinate values.
CartesianCoordinates	This class is a subclass of the PhysicalCoordinates class and implements a coordinate system used to plot linear, logarithmic and semi-logarithmic graphs.
TimeCoordinates	This class is a subclass of the PhysicalCoordinates class and implements a coordinate system used to plot GregorianCalendar time-based data.
Attribute Class ChartAttribute	

This class consolidates the common line and fill attributes as a single class. Most of the graph objects have a property of this class that controls the color, line thickness and fill attributes of the object.

ChartAttribute This class consolidates the common line and fill attributes associated with a **GraphObj** object into a single class.

Auto-Scaling Classes

AutoScale

LinearAutoScale

LogAutoScale

TimeAutoScale

Usually, programmers do not know in advance the scale for a chart. Normally the program needs to analyze the current data for minimum and maximum values and create a chart scale based on those values. Auto-scaling, and the creation of appropriate axes, with endpoints at even values, and well-rounded major and minor tick mark spacing, is quite complicated. The **AutoScale** classes provide tools that make automatic generation of charts easier.

AutoScale	This class is the abstract base class for the auto-scale classes.
LinearAutoScale	This calculates scaling values based on the numeric values in SimpleDataset , SimpleDataset3D , GroupDataset and GroupDataset3D objects. Linear scales and axes use it for auto-scale calculations.
LogAutoScale	This class calculates scaling values based on the numeric values in SimpleDataset , GroupDataset and GroupDataset3D objects. Logarithmic scales and axes use it for auto-scale calculations.
TimeAutoScale	This calculates scaling values based on the ChartCalendar values in TimeSimpleDataset , TimeSimpleDataset3D , TimeGroupDataset and TimeGroupDataset3D objects. Date/time scales and axes use it for auto-scale calculations.

Chart Object Classes

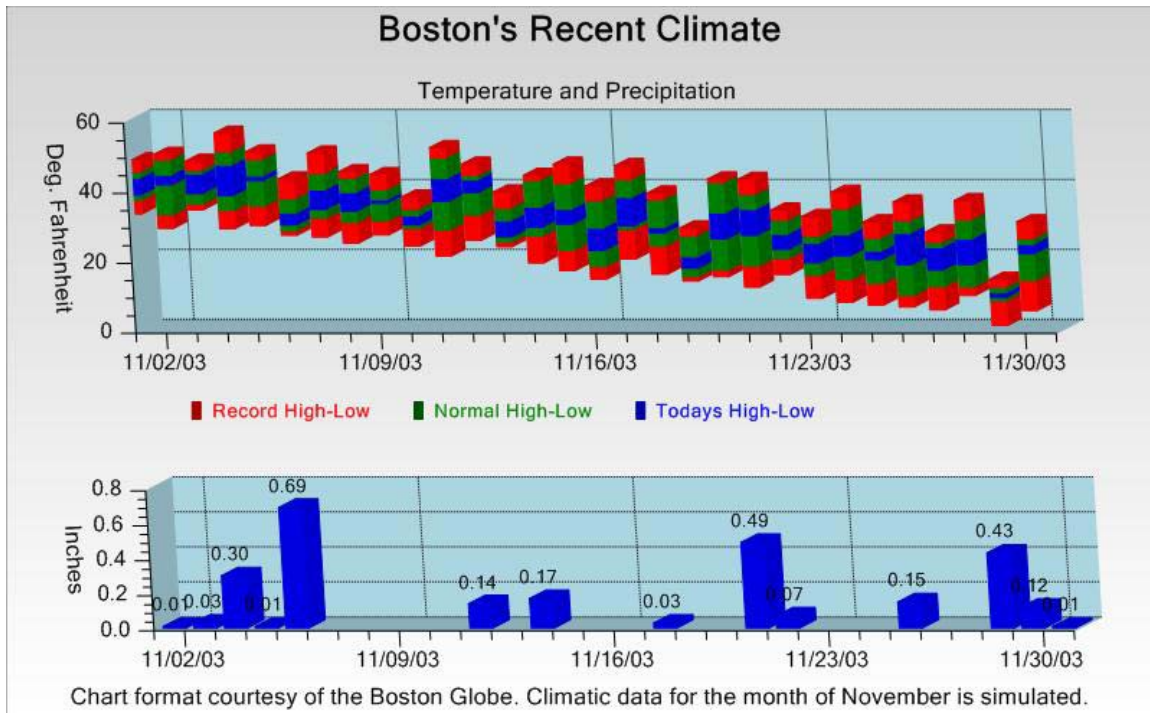
Chart objects are graph objects that can be rendered in the current graph window. This is in comparison to other classes that are purely calculation classes, such as the coordinate conversion classes. All chart objects have certain information in common. This includes instances of **ChartAttribute** and **PhysicalCoordinates** classes. The **ChartAttribute** class contains basic color and line style information for the object, while the **PhysicalCoordinates** maintains the coordinate system used by object. The majority of classes in the library derive from the **GraphObj** class, each class a specific charting object such as an axis, an axis label, a simple plot or a group plot. Add **GraphObj** derived objects (axes, plots, labels, title, etc.) to a graph using the **ChartView.AddChartObject** method.

GraphObj

This class is the abstract base class for all drawable graph objects. It contains information common to all chart objects. This class includes references to instances of the **ChartAttribute** and **PhysicalCoordinates** classes. The **ChartAttribute** class contains basic color and line style information for the object, while the **PhysicalCoordinates** maintains the coordinate system used by object. The majority of classes in the library derive from the **GraphObj** class, each class a specific charting object such as an axis, an axis label, a simple plot or a group plot

Background

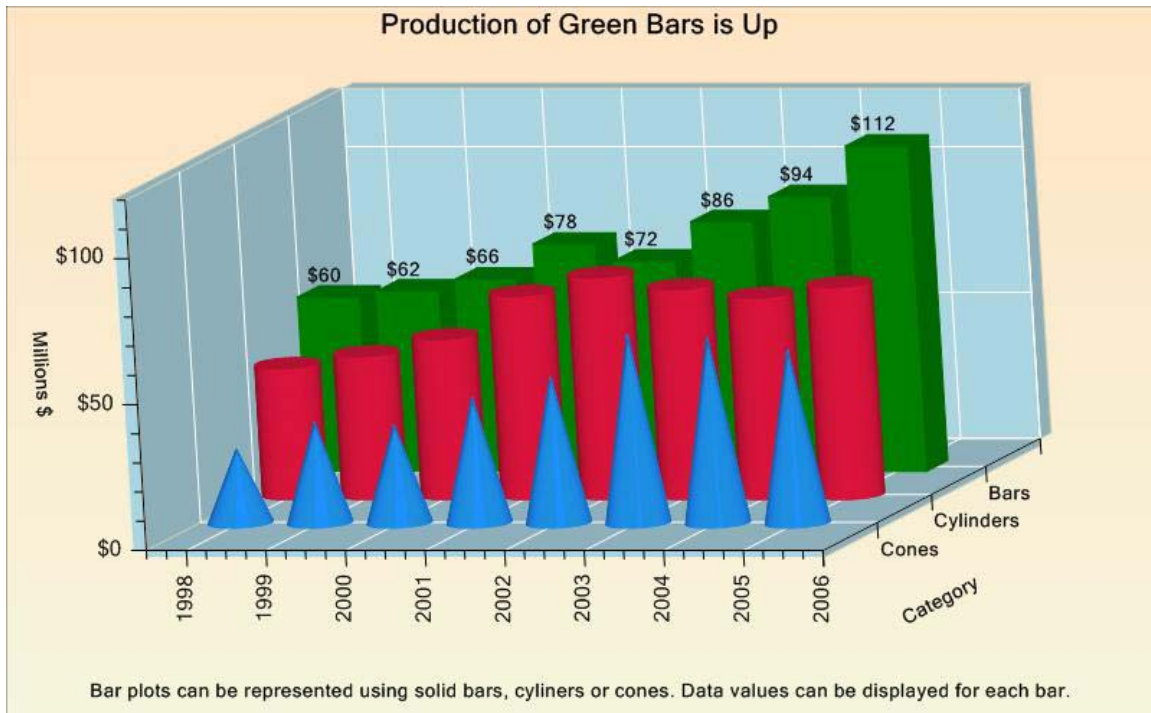
This class fills the background of the entire chart using a solid color, a color gradient, or a texture. It can also be used to fill the plot area (the area bounded by the charts 3D coordinate system) using simple wall-like planes of a solid color.



This example combines a graph area background that uses a gradient (LightGray to White), with two different plot area background objects (LightBlue) (Bargraphs.BostonClimate)

Wall3D

This class defines walls that can be used as background elements of a 3D plot. Unlike the simple plane walls created using the Background class, the **Wall3D** class defines walls as true 3D objects with height, width and depth.



This example combines a graph area background with three Wall3D objects (LightBlue) (Bargraphs.SimpleBars)

Axis Classes

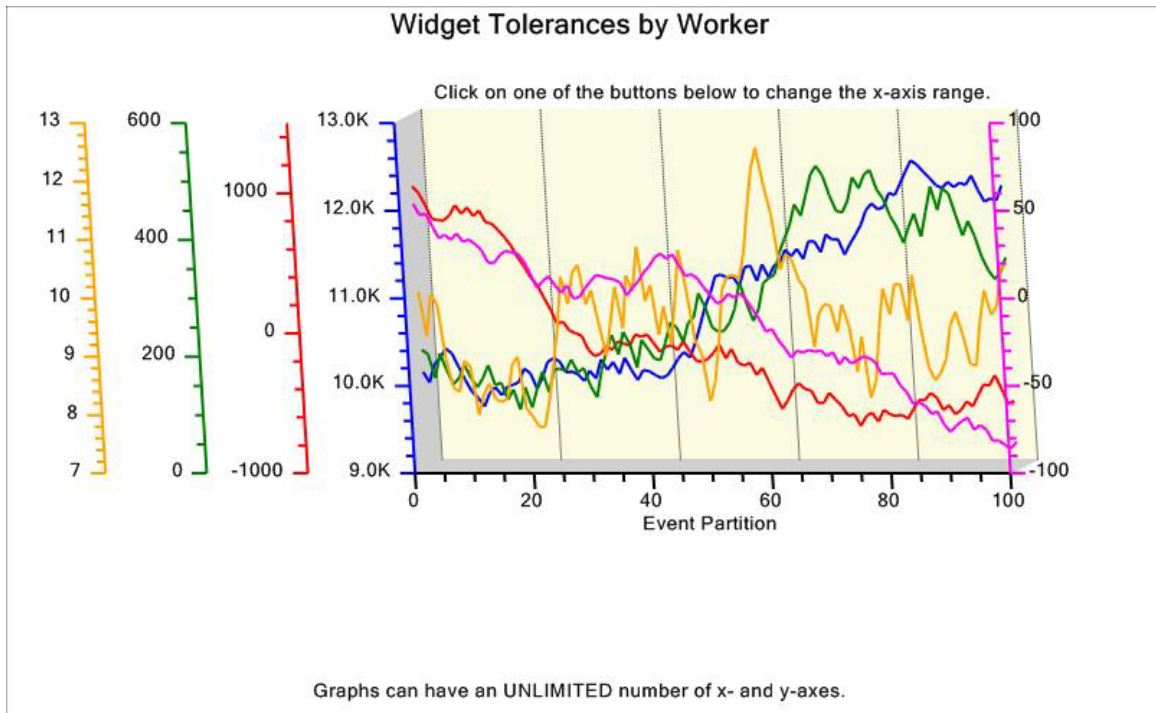
Axis

LinearAxis

LogAxis

TimeAxis

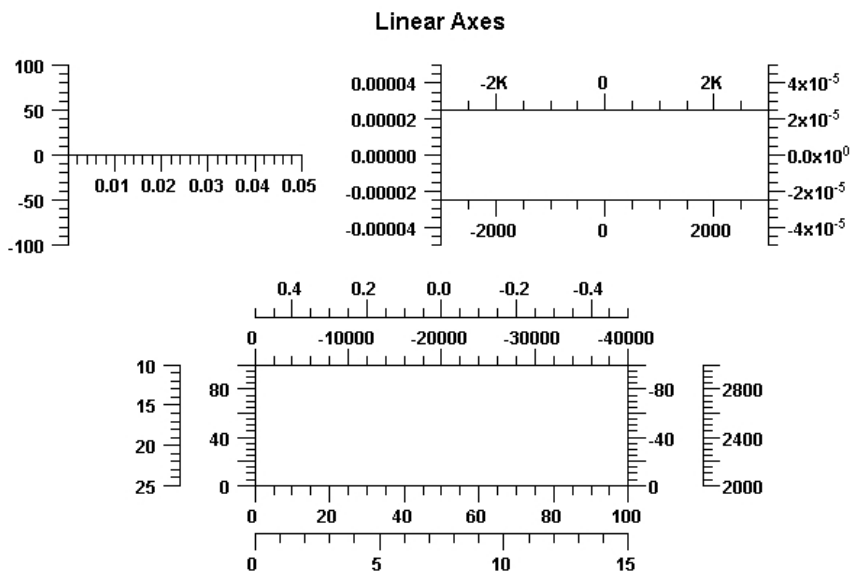
Creating a **PhysicalCoordinates** coordinate system does not automatically create a set of x-, y-, and z-axes. Axes are separate charting objects drawn with respect to a specific **PhysicalCoordinates** object. The coordinate system and the axes do not need to have the same limits. In general, the limits of the coordinate system should be greater than or equal to the limits of the axes. The coordinate system may have limits of 0 to 15, while you may want the axes to extend from 0 to 10.



Example program `MultipleAxes.MultiAxesChart`

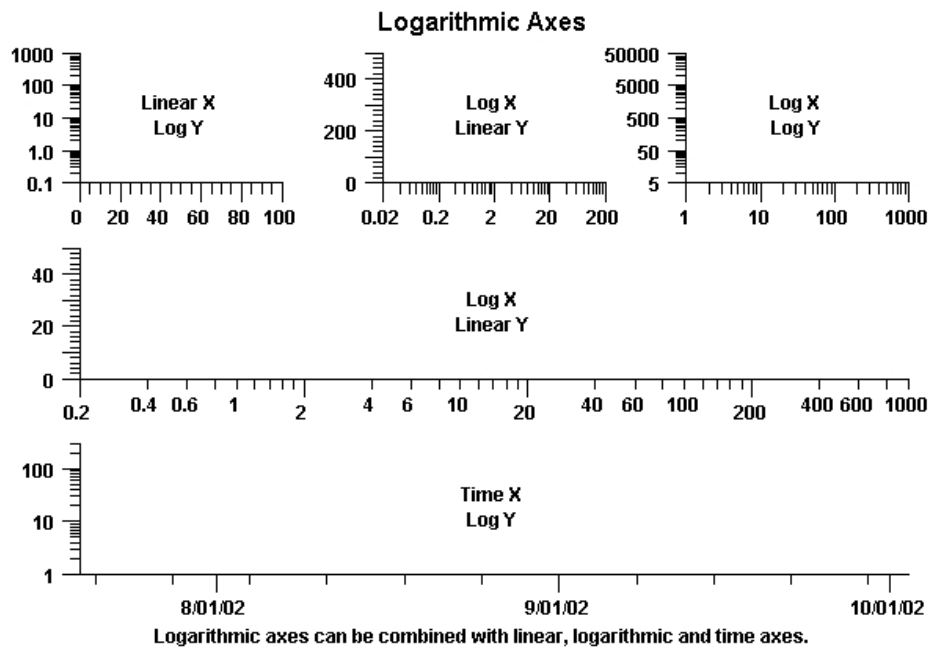
Axis

This class is the abstract base class for the other axis classes. It contains data and drawing routines common to all axis classes.



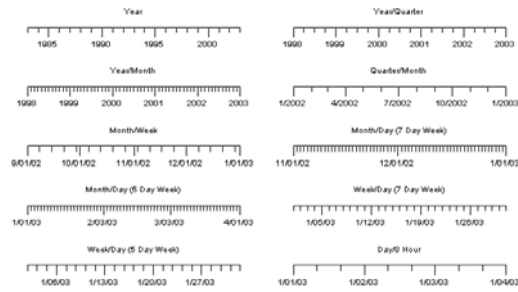
LinearAxis

This class implements a linear axis with major and minor tick marks placed at equally spaced intervals.

**LogAxis**

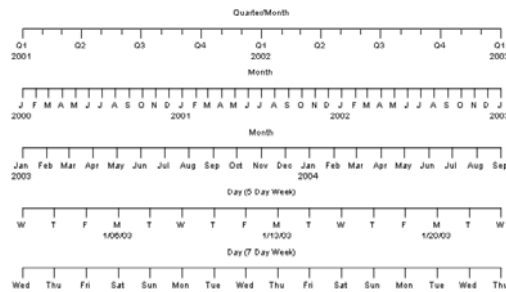
This class implements a logarithmic axis with major tick marks placed on logarithmic intervals, for example 1, 10, 100 or 30, 300, 3000. The minor tick marks are placed within the major tick marks using linear intervals, for example 2, 3, 4, 5, 6, 7, 8, 9, 20, 30, 40, 50,..., 90. An important feature of the **LogAxis** class is that the major and minor tick marks do not have to fall on decade boundaries. A logarithmic axis must have a positive range exclusive of 0.0, and the tick marks can represent any logarithmic scale.

Date Axes



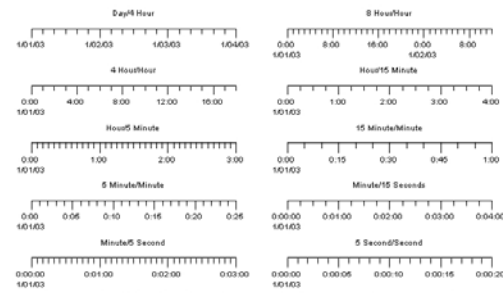
There are more than 40 different time/date axes types, appropriate for scales ranging from 1 second to 100 years.

Date Axes



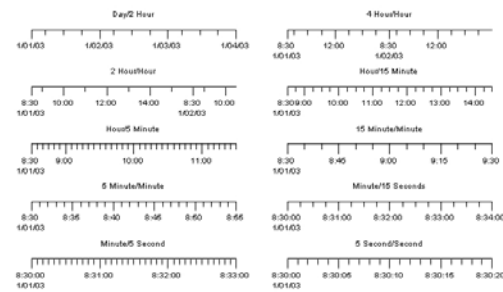
There are more than 40 different time/date axes types, appropriate for scales ranging from 1 second to 100 years.

Date Axes



There are more than 40 different time/date axes types, appropriate for scales ranging from 1 second to 100 years.

Time/Date Axes with Custom Hour Range (8:30 AM to 4:00 PM)



A time axis can be customized for non-24 hour range.

TimeAxis

This class is the most complex of the axis classes. It supports time scales ranging from 1 millisecond to hundreds of years. Dates and times are specified using the **ChartCalendar** class. The major and minor tick marks can fall on any time base, where a time base represents seconds, minutes, hours, days, weeks, months or years. The scale can exclude weekends, for example, Friday, October 20, 2000 is immediately followed by Monday, October 23, 2000. A day can also have a custom range, for example a range of 9:30 AM to 4:00 PM. The chart time axis excludes time outside of this range. This makes the class very useful for the inter-day display of financial market information (stock, bonds, commodities, options, etc.) across several days, months or years.

Axis Label Classes

AxisLabels

NumericAxisLabels

StringAxisLabels

TimeAxisLabels

Axis labels inform the user of the x-, y-, and z-scales used in the chart. The labels center on the major tick marks of the associated axis. Axis labels are usually numbers, times, dates, or arbitrary strings.

Axis Labels

Possible date labels for todays date	Possible time labels for the current time	Possible numeric labels for the value 12340	
July 19, 2002	15:15:11 (24 Hour Mode)	12340.0	(Decimal)
2002	15:15 (24 Hour Mode)	1.2340E4	(Scientific)
7/2002	15:11 Minute:Second	12.340K	(Business)
7/19/2002	3:15:11 (12 Hour Mode)	1234000%	(Percent)
19/07/2002	3:15 (12 Hour Mode)	1.2340x10 ⁴	(Exponent)
02		\$12340	(Currency)
7/02			
7/19/02			
19/07/02			
July			
Jul			
J			
Friday			
Fri			
F			

In addition to the predefined formats, programmers can define custom time, date and numeric formats.

AxisLabels

This class is the abstract base class for all axis label objects. It places numeric labels, date/time labels, or arbitrary text labels, at the major tick marks of the associated axis object. In addition to the standard font options (type, size, style, color, etc.), axis label text can be rotated 360 degrees in one degree increments.

NumericAxisLabels

This class labels the major tick marks of the **LinearAxis**, and **LogAxis** classes. The class supports many predefined and user-definable formats, including numeric, exponent, percentage, business and currency formats.

StringAxisLabels

This class labels the major tick marks of the **LinearAxis**, and **LogAxis** classes using user-defined strings.

TimeAxisLabels

This class labels the major tick marks of the associated **TimeAxis** object. The class supports many time (23:59:59) and date (5/17/2001) formats. It is also possible to define custom date/time formats.

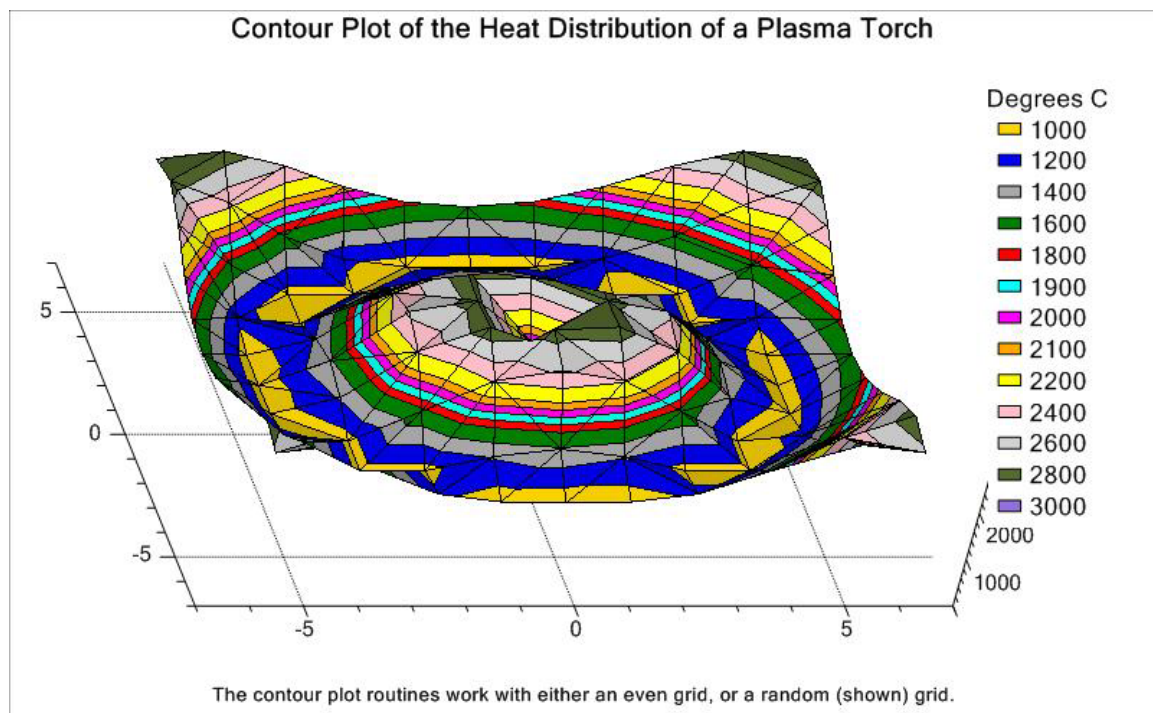
Chart Plot Classes

ChartPlot**ContourPlot****GroupPlot****PieChart****SimplePlot**

Plot objects are objects that display data organized in a **ChartDataset** class. There are five main categories: simple, group, contour and pie plots. Simple plots graph data organized as a simple set of x,y and z-values. The z-values can be a fixed constant, as in the case of 2½ D plots, or can vary from point to points, as in the case of true 3D plots. The most common examples of simple plots are line plots, bar graphs and scatter plots. Group plots graph data organized as multiple y-values for each x-value. Most group plots do not require an independent set of z-values, so in most group plots the z-value is always set to a constant. The most common examples of group plots are stacked and group bar graphs, open-high-low-close plots, and candlestick plots. The contour plot type displays the iso-lines, or contours, of a 3D surface using either lines or regions of solid color. The last plot object category is the pie chart, where a pie wedge represents each data value. The size of the pie wedge is proportional to the fraction (data value / sum of all data values).

ChartPlot

This class is the abstract base class for chart plot objects. It contains a reference to a **ChartDataset** derived class containing the data associated with the plot.



Contour fill plot (ContourPlots.ContourFillPlot)

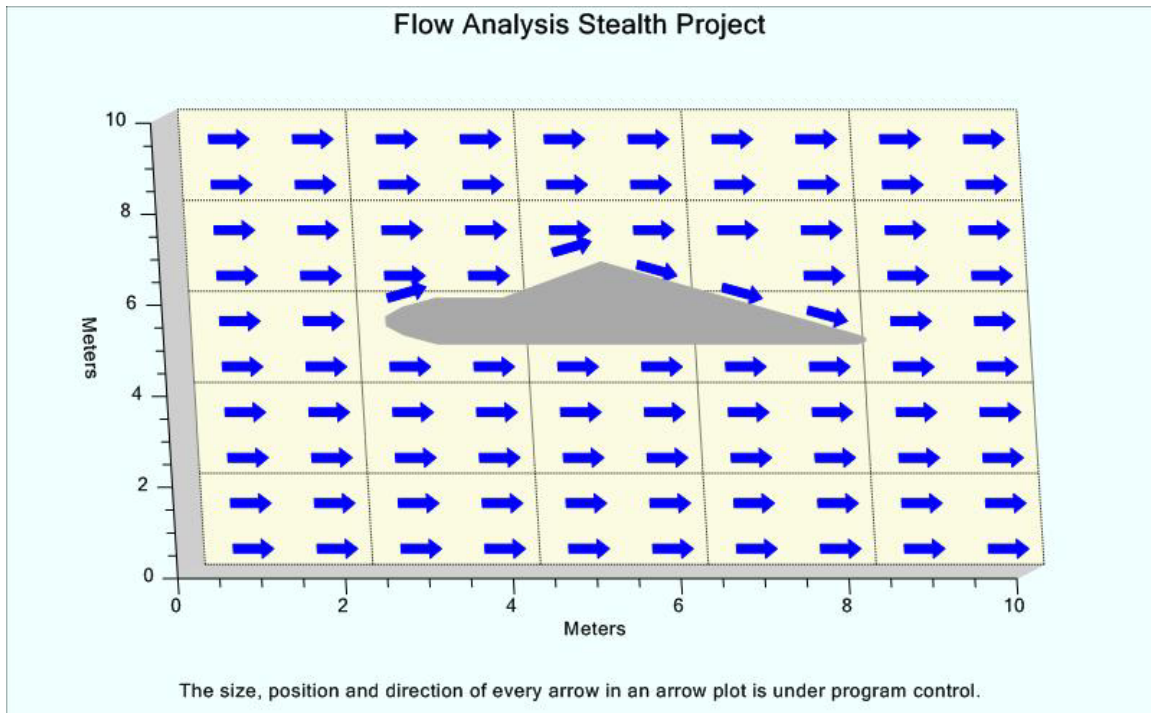
ContourPlot This class is a subclass of the **ChartPlot** class and displays a contour plot using either lines, or regions filled with color.

Group Plot Classes**GroupPlot**

ArrowPlot
BubblePlot
CandlestickPlot
CellPlot
ErrorBarPlot
FloatingBarPlot
FloatingStackedBar
GroupBarPlot
HistogramPlot
LineGapPlot
MultiLinePlot
MultiRibbonPlot
OHLCPlot
OHLCBarPlot
StackedBarPlot
StackedLinePlot
StackedRibbonPlot

Group plots use data organized as arrays of x- and y-values, where there is one or more y for every x. Group plot types include multi-line plots, stacked line plots, stacked bar plots, group bar plots, error bar plots, floating bar plots, open-high-low-close plots, candlestick plots, arrow plots, histogram plots, cell plots and bubble plots.

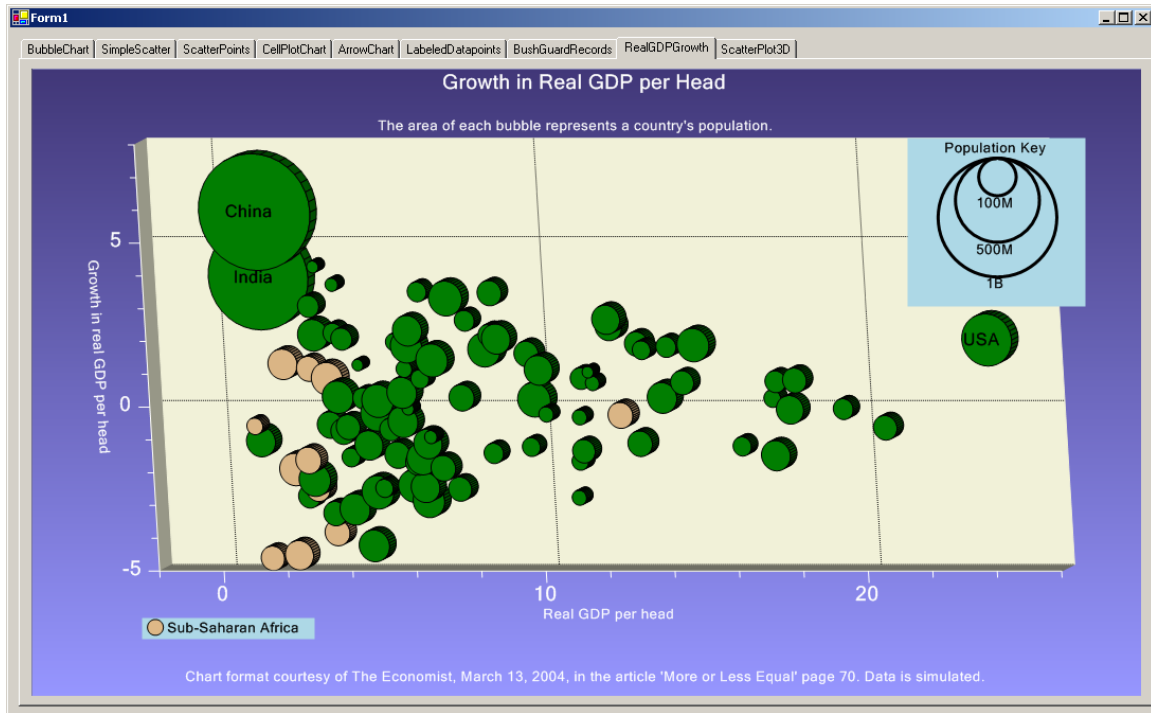
GroupPlot This class is an abstract base class for all group plot classes.



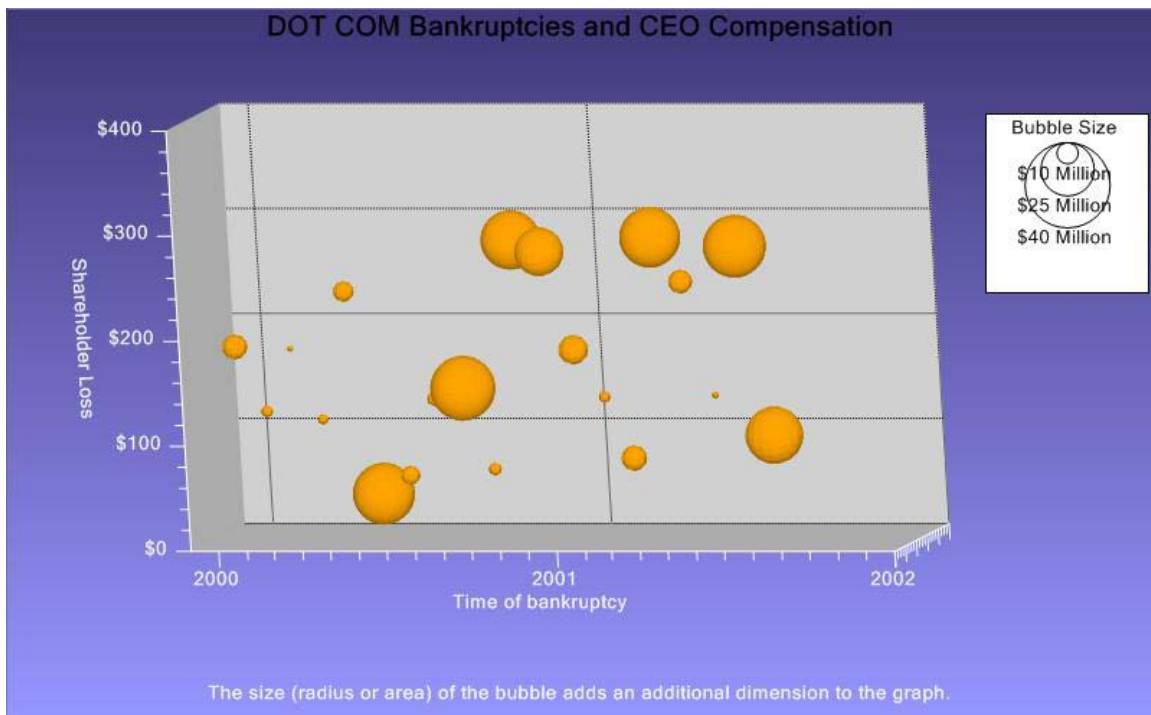
Arrow plot (ScatterPlots.ArrowChart)

ArrowPlot

This displays a collection of arrows as defined by the data in a group dataset. The position, size, and rotation of each arrow in the collection is independently controlled.



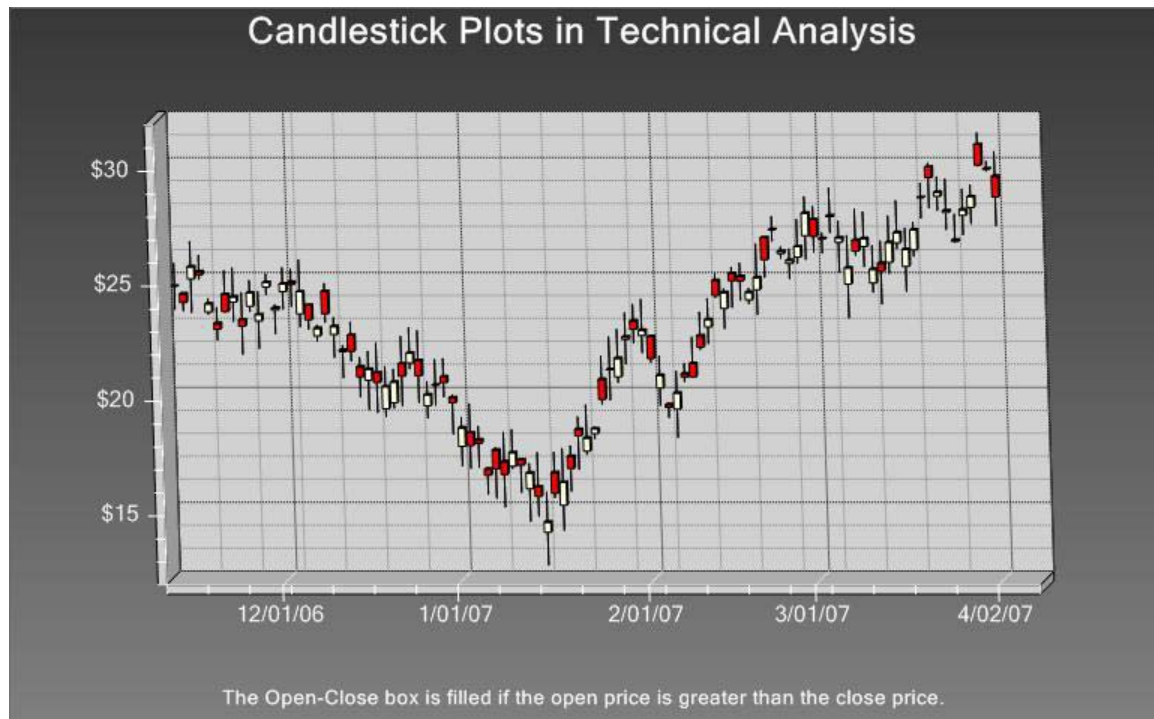
Bubble plot: disk mode. Example program `ScatterPlots.RealGDPGrowth`



Bubble plot: sphere mode. (`ScatterPlots.BubbleChart`)

BubblePlot

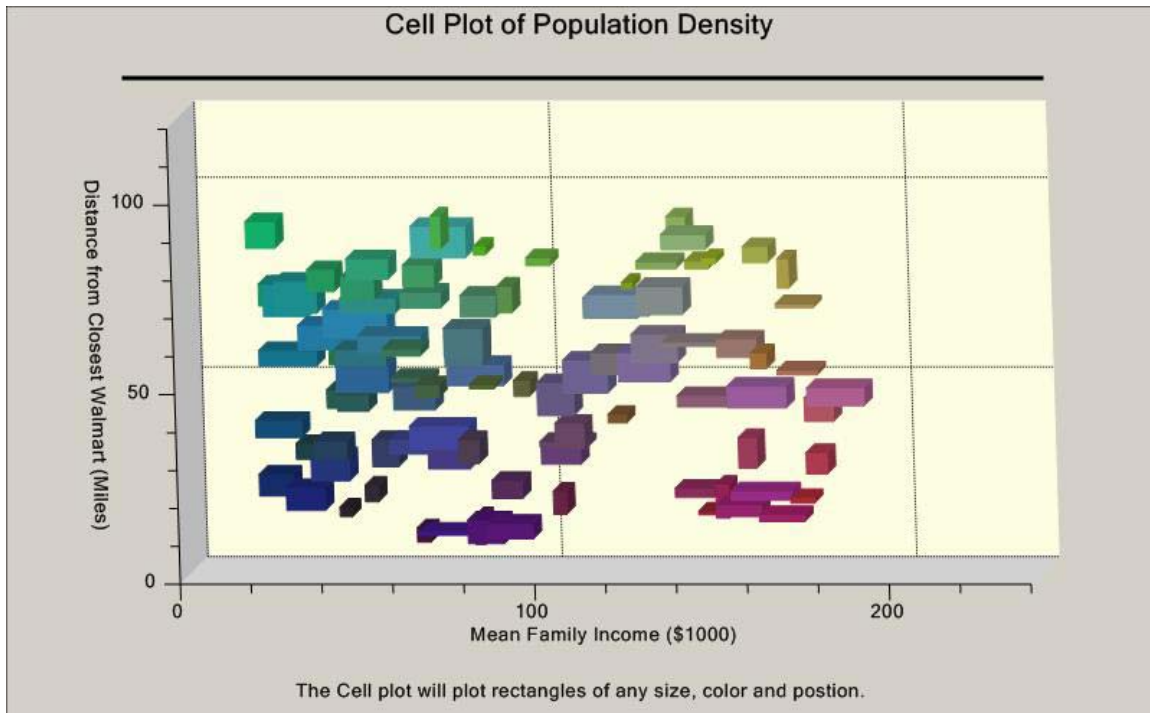
This displays bubble plots as either disks, or spheres in a 3D view. The **BubblePlot** class can plot the bubbles as a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property, or as a 3D chart, where the z-value for every bubble is set independently. The values in the dataset specify the position and size of each bubble in a bubble chart.



Candlestick plot (FinancialExamples.CandlestickChart)

CandlestickPlot

This displays stock market data in an open-high-low-close format common in financial technical analysis. The **CandlestickPlot** class can plot the candlestick symbols as a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property, or as a 3D chart, where the z-value for every candlestick symbols is set independently. The depth of the central 3D rectangle is set using the associated datasets **ImplicitDepthValue** property.



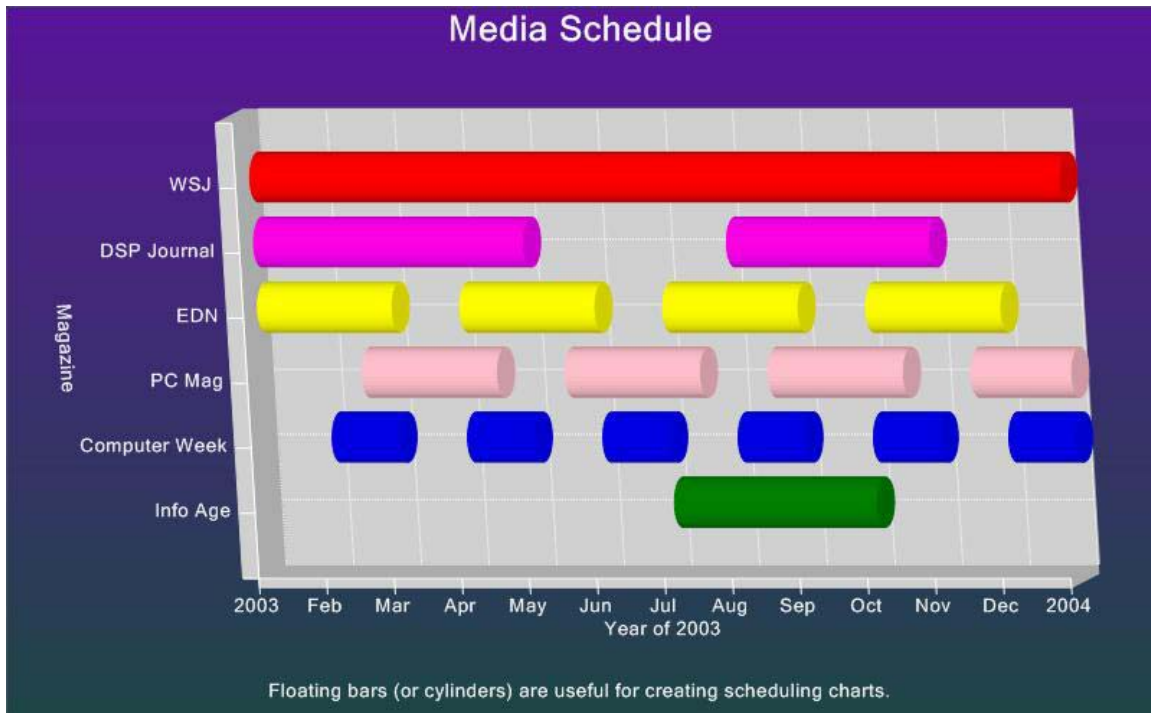
Cell plot (ScatterPlots.CellPlotChart)

CellPlot

This class displays a collection of rectangular objects with independent positions, widths and heights, specified using the values of the associated group dataset. The **CellPlot** class can plot the cells as a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property, or as a 3D chart, where the z-value for every cell is set independently. The depth of the 3D rectangle is set using the associated datasets **ImplicitDepthValue** property.

ErrorBarPlot

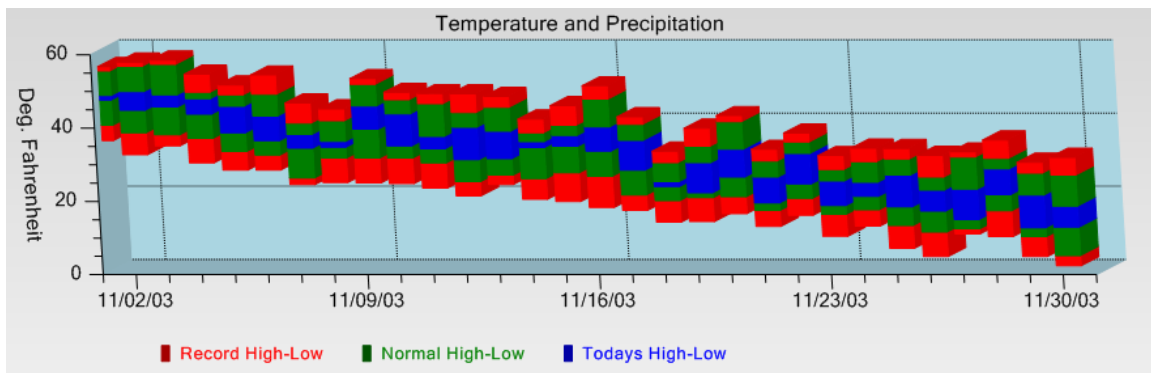
This class displays group data as error bars. Error bars are two parallel planes positioned about a data point that signify the statistical error associated with the data point. The **ErrorBarPlot** class can plot the error bars as a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property, or as a 3D chart, where the z-value for every set of error bars is set independently. The depth of the planes is set using the associated datasets **ImplicitDepthValue** property.



Floating bar plot using the cylinder option (Bargraphs.FloatingBars)

FloatingBarPlot

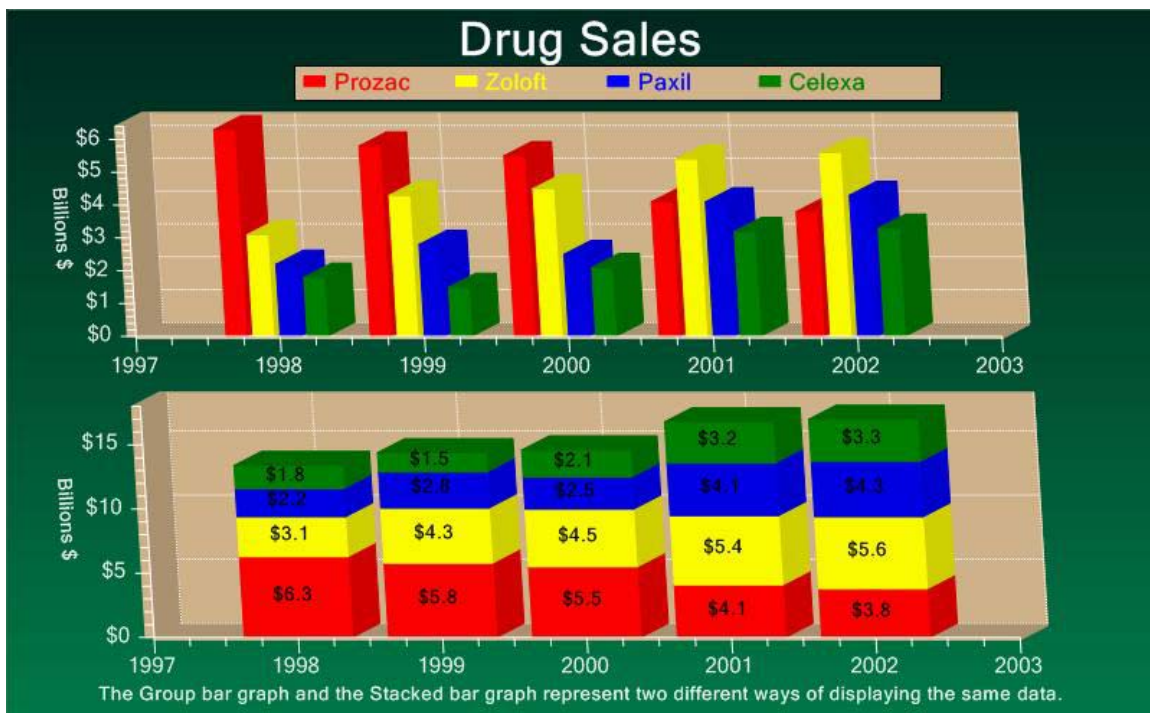
This class displays free-floating bars and cylinders in a graph. The bars and cylinders are free floating because each bar does not reference a fixed base value, as do simple bar plots, stacked bar plots and group bar plots. The **FloatingBarPlot** class can plot the bars as a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property, or as a 3D chart, where the z-value for every bar is set independently. The depth of the 3D rectangle is set using the associated datasets **ImplicitDepthValue** property.



Floating stacked bar plot (Bargraphs.BostonClimate)

FloatingStackedBarPlot

This class displays stacked floating bar plots. In a stacked floating bar plot each group is stacked on top of one another, each group bar a cumulative sum of the related group items before it. The bars are free floating because each bar does not reference a fixed base value, as do stacked bars. All bars in a given **FloatingStackedBarPlot** object have the same width. The **FloatingStackedBarPlot** class can plot the bars as a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property, or as a 3D chart, where the z-value for every bar is set independently.



Group (top) and stacked (bottom) bar plots (Bargraphs.GroupBargraphs)

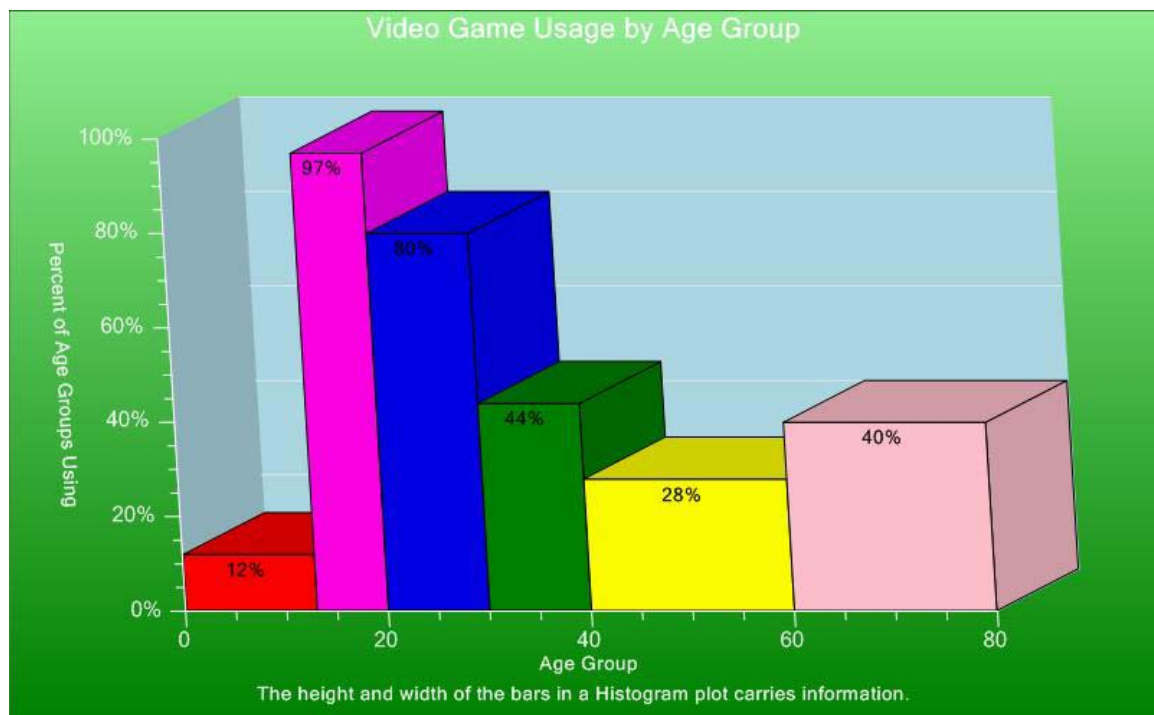
GroupBarPlot

This class displays group data in a group bar format. Individual bars, the height of which corresponds to the group y-values of the dataset, display side by side, as a group, justified with respect to the x-position value for each group. The group bars share a common base value. The **GroupBarPlot** class can plot the bars as a 2½ D chart, where the z-value of the plot is fixed using the associated

datasets **ImplicitZValue** property, or as a 3D chart, where the z-value for every bar is set independently. The depth of the 3D rectangle is set using the associated datasets **ImplicitDepthValue** property.

StackedBarPlot

This class displays group data as stacked bars. In a stacked bar plot each group is stacked on top of one another, each group bar a cumulative sum of the related group items before it. The **StackedBarPlot** class can plot the bars as a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property, or as a 3D chart, where the z-value for every bar is set independently. The depth of the 3D rectangle is set using the associated datasets **ImplicitDepthValue** property.

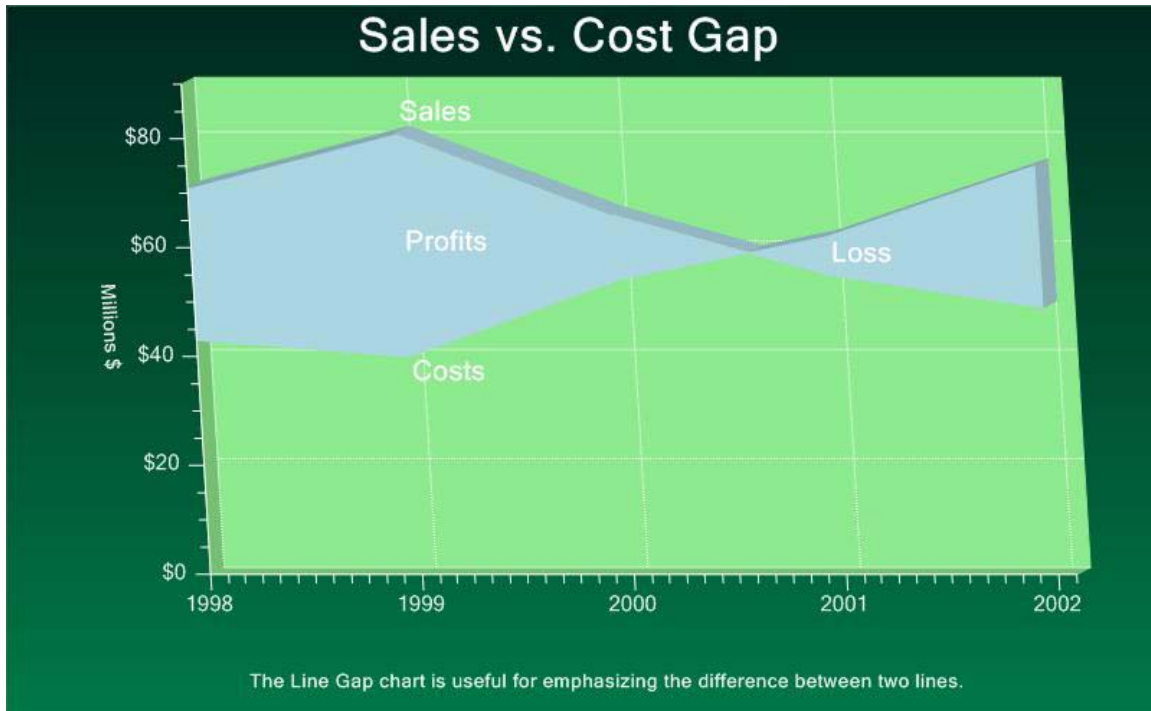


Histogram plots (Bargraphs.HistogramBars)

HistogramPlot

This class displays group data as a histogram plots. A histogram plot is a collection of rectangular objects with independent widths and heights, specified using the values of the associated group dataset. The histogram bars share a common base value. The **HistogramPlot** class can plot the bars as a 2½ D chart, where the z-value of the plot is fixed

using the associated datasets **ImplicitZValue** property, or as a 3D chart, where the z-value for every bar is set independently. The depth of the 3D rectangle is set using the associated datasets **ImplicitDepthValue** property.

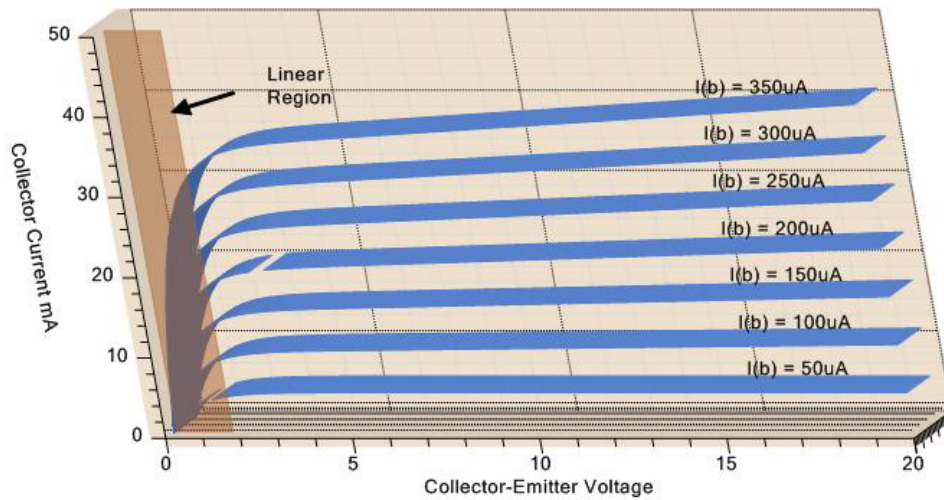


Line gap plot (MiscCharts.LineGap)

LineGapPlot

This class plots group data as a line gap chart. The plot consists of two lines plots where a contrasting color fills the area between the two lines, highlighting the difference. The **LineGapPlot** class is a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property. . The depth of the plot is set using the associated datasets **ImplicitDepthValue** property.

Characteristic Curves for Bipolar Transistor

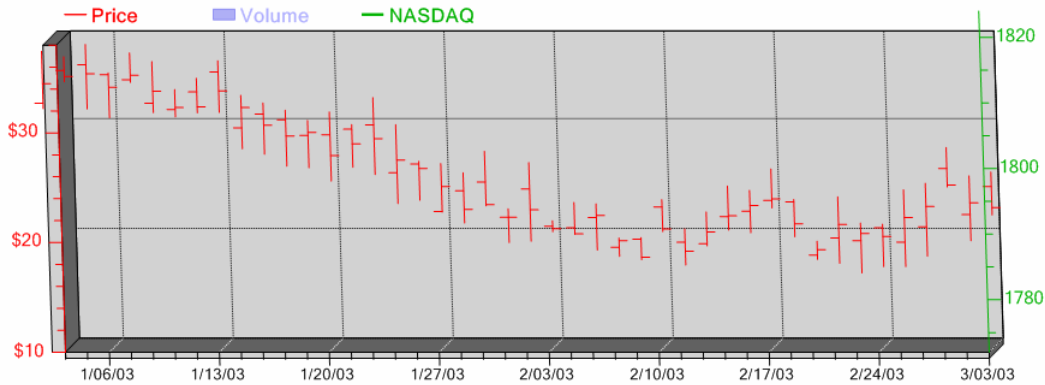


The `MultiLinePlot` will plot a multiple y-vectors vs a single x-vector.

Multi-ribbon plots (`MultiLinePlots.MultiLines1`)

MultiRibbonPlot

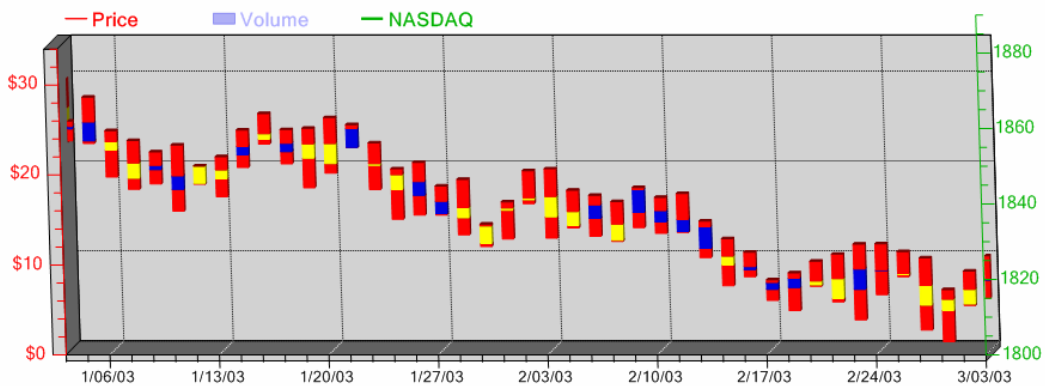
This class displays group data in multi-line format. A group dataset with four groups will display four separate line plots. The y-values for each line of the line plot represent the y-values for each group of the group dataset. Each line plot share the same x-values of the group dataset. The **MultiRibbonPlot** class is a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property. The depth of the ribbons are set using the associated datasets **ImplicitDepthValue** property.



Open-High-Low-Close (OHLC) plot (FinancialExamples.OHLCChart)

OHLCPlot

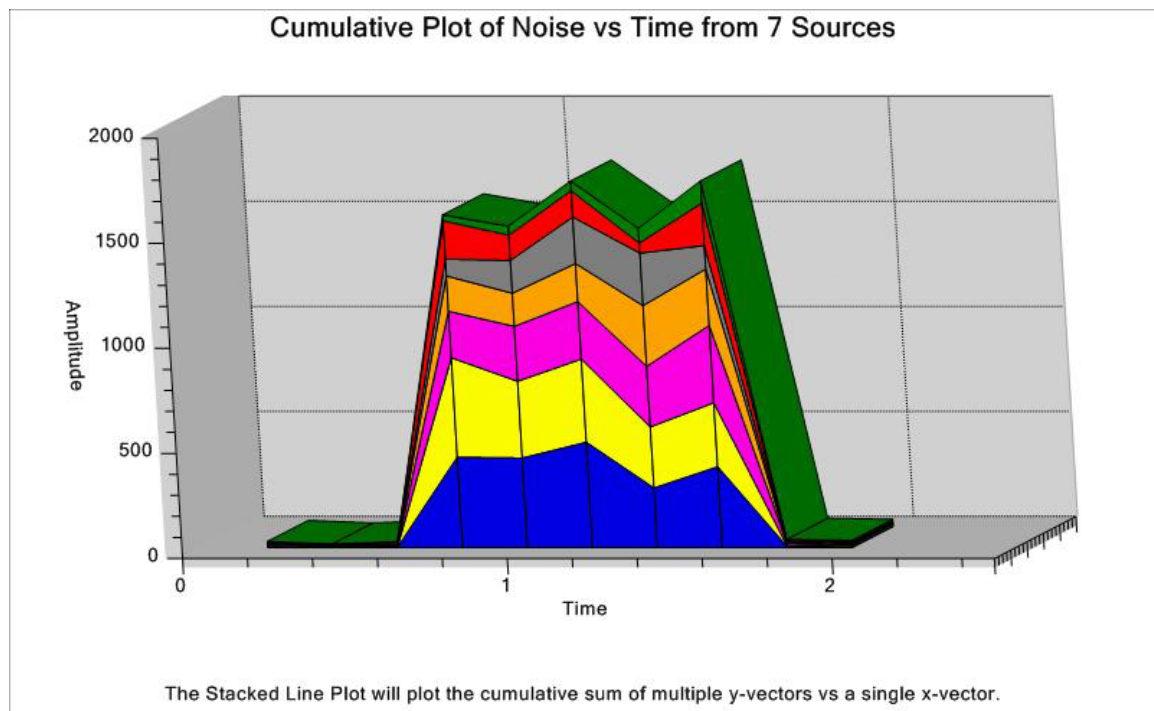
This class displays stock market data in an open-high-low-close format common in financial technical analysis. In the **OHLCPlot** class, every item of the plot is a vertical line, representing High and Low values, with two small horizontal "flags", one left and one right extending from the vertical High-Low line and representing the Open and Close values. The **OHLCPlot** class can plot the bars as a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property, or as a 3D chart, where the z-value for every bar is set independently.



Ooen-High-Low-Close (OHLC) bar plot (FinancialExamples.OHLCChart)

OHLCBarPlot

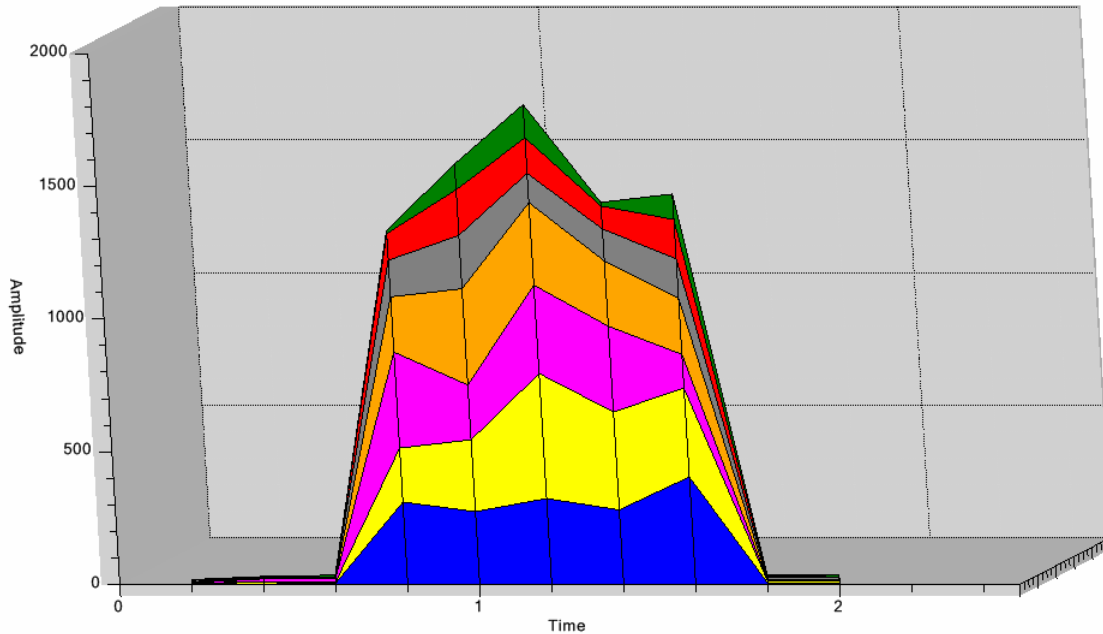
This class displays stock market data in an open-high-low-close format common in financial technical analysis. In the **OHLCBarPlot** class, 3D bars with differentiating colors are used to display the open-high-low-close values. The **OHLCBarPlot** class can plot the bars as a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property, or as a 3D chart, where the z-value for every bar is set independently. The depth of the 3D bars are set using the associated datasets **ImplicitDepthValue** property.



Stacked ribbon plots (MultiLinePlots.StackedLines)

StackedRibbonPlot

This class displays data in a stacked ribbon format. In a stacked ribbon plot each group is stacked on top of one another, each group line a cumulative sum of the related group items before it. The **StackedRibbonPlot** class is a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property. The depth of the ribbons are set using the associated datasets **ImplicitDepthValue** property.



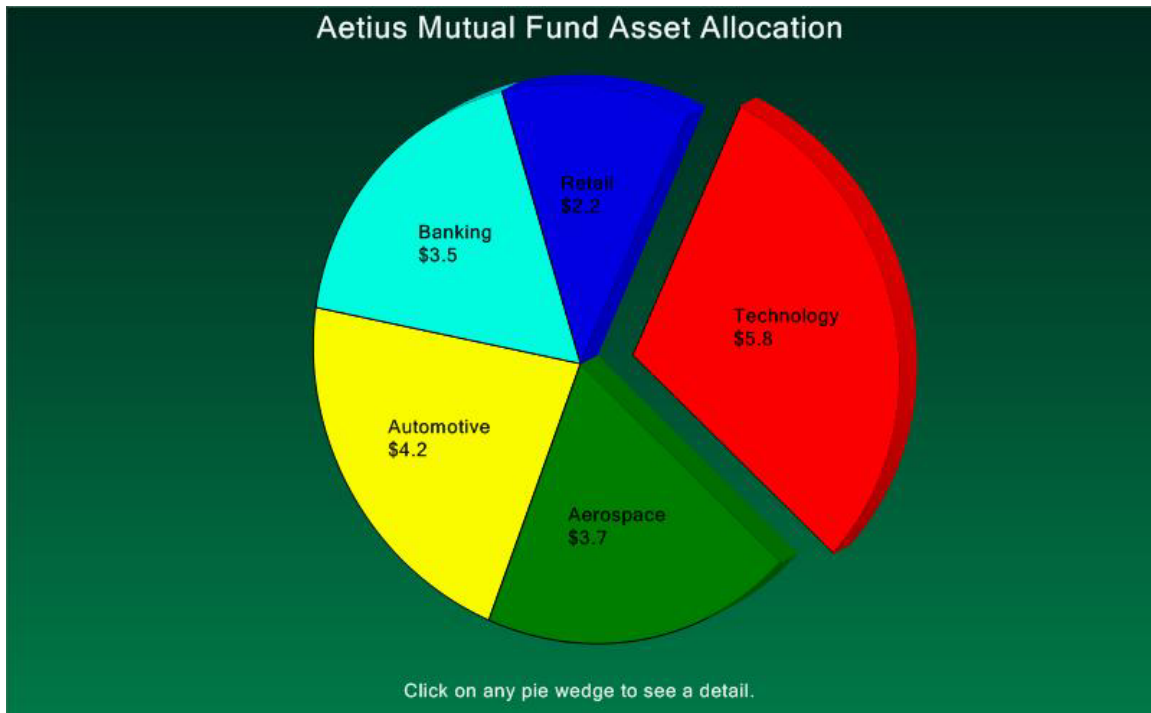
Stacked line plots (`MultiLinePlots.StackedLines`)

StackedLinePlot

This class displays data in a stacked line format. In a stacked line plot each group is stacked on top of one another, each group line a cumulative sum of the related group items before it. The **StackedLinePlot** class is a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property.

Pie Chart Classes

It uses data organized as arrays of x- and y-values, where an x-value represents the numeric value of a pie wedge, and a y-value specifies the offset (or “explosion”) of a pie wedge with respect to the center of the pie.



Pie Chart (PieCharts.SimplePieChart)

PieChart

This class plots data in a simple pie chart format. It uses data organized as arrays of x- and y-values, where an x-value represents the numeric value of a pie wedge, and a y-value specifies the offset (or “explosion”) of a pie wedge with respect to the center of the pie. The depth of the pie is set using the associated datasets **ImplicitDepthValue** property.

Simple Plot Classes

SimplePlot

SimpleBarPlot

SimpleLineMarkerPlot

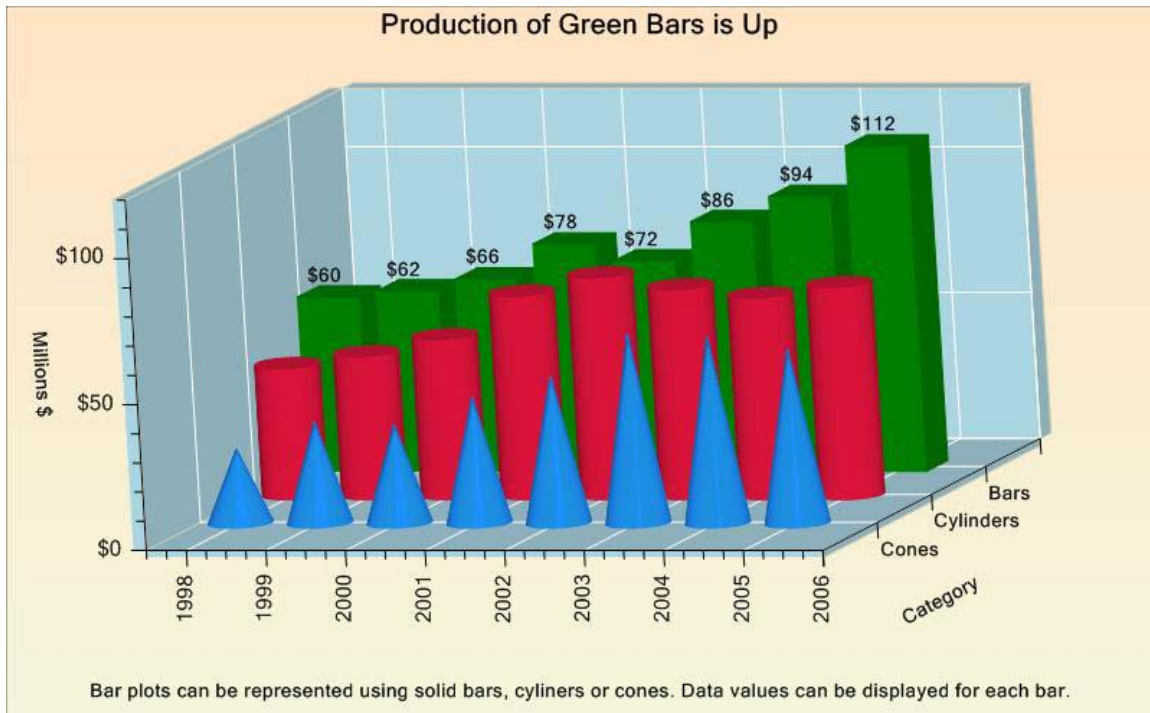
SimpleLinePlot

SimpleScatterPlot

Simple plots use data organized as a simple array of xy points, where there is one y for every x. Simple plot types include line plots, scatter plots, bar graphs, and line-marker plots.

SimplePlot

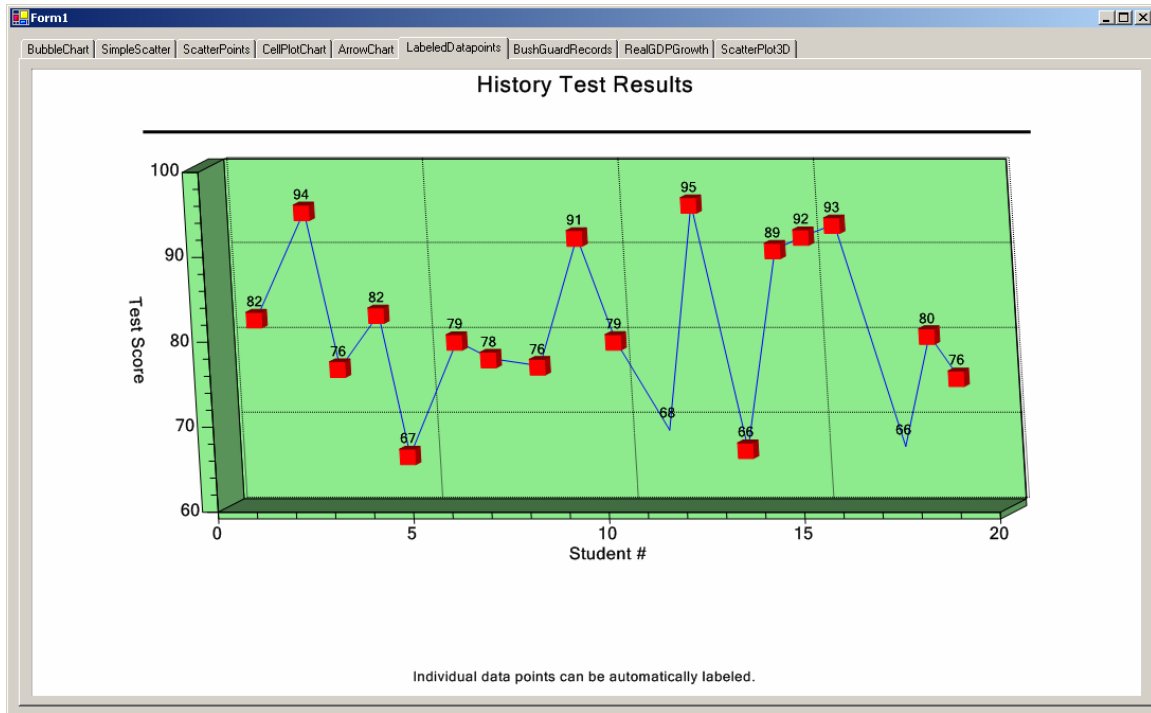
This class is an abstract base class for all simple plot classes.



Simple bar plots (Bargraphs.SimpleBars)

SimpleBarPlot

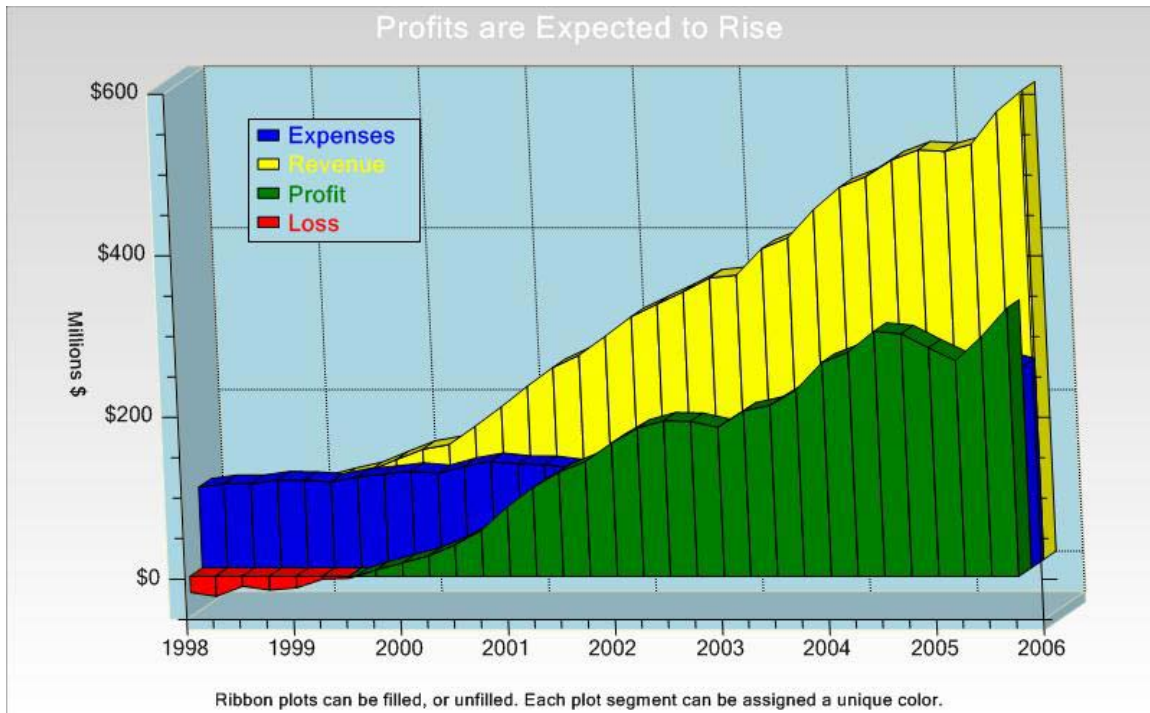
This class displays data in a bar format. Individual bars, the maximum value of which corresponds to the y-values of the dataset, are justified with respect to the x-values. Cylinders and cones can be displayed instead of bars. When combined with 2½ D data, the z-value of the bars is fixed using the associated datasets **ImplicitZValue** property. When used with 3D data the z-value of each bar is set using the datasets associated z-data value for that point. The depth of the bars is set using the datasets **ImplicitDepthValue** property.



Labeled data points (ScatterPlots.LabeledDatapoints)

SimpleLineMarkerPlot

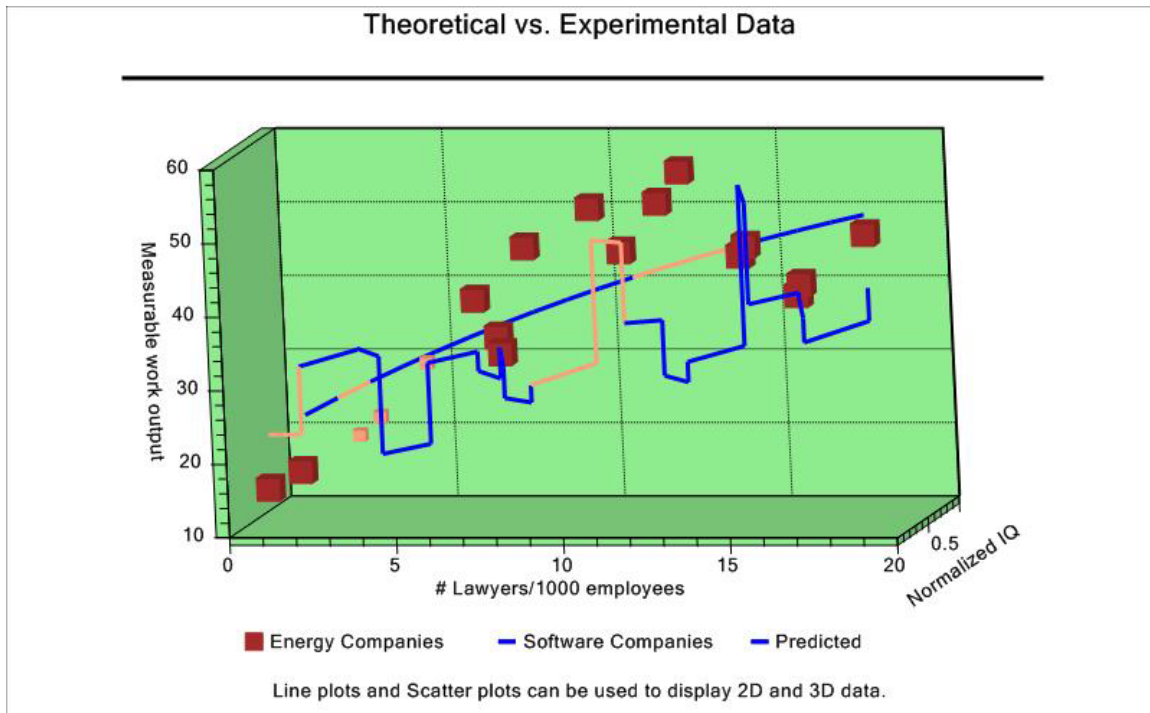
This class is a subclass of the **SimplePlot** class and it displays simple datasets in a line plot format where scatter plot symbols highlight individual data points. Each data point is represented using a 2D or 3D scatter plot symbols. The class will work with 2½ D data, using a constant z-value, set using the associated datasets **ImplicitZValue** property, or 3D data, where each data point is represented using individual x, y and z datapoints.



Filled ribbon plot (SimpleLinePlots.LineFill)

SimpleRibbonPlot

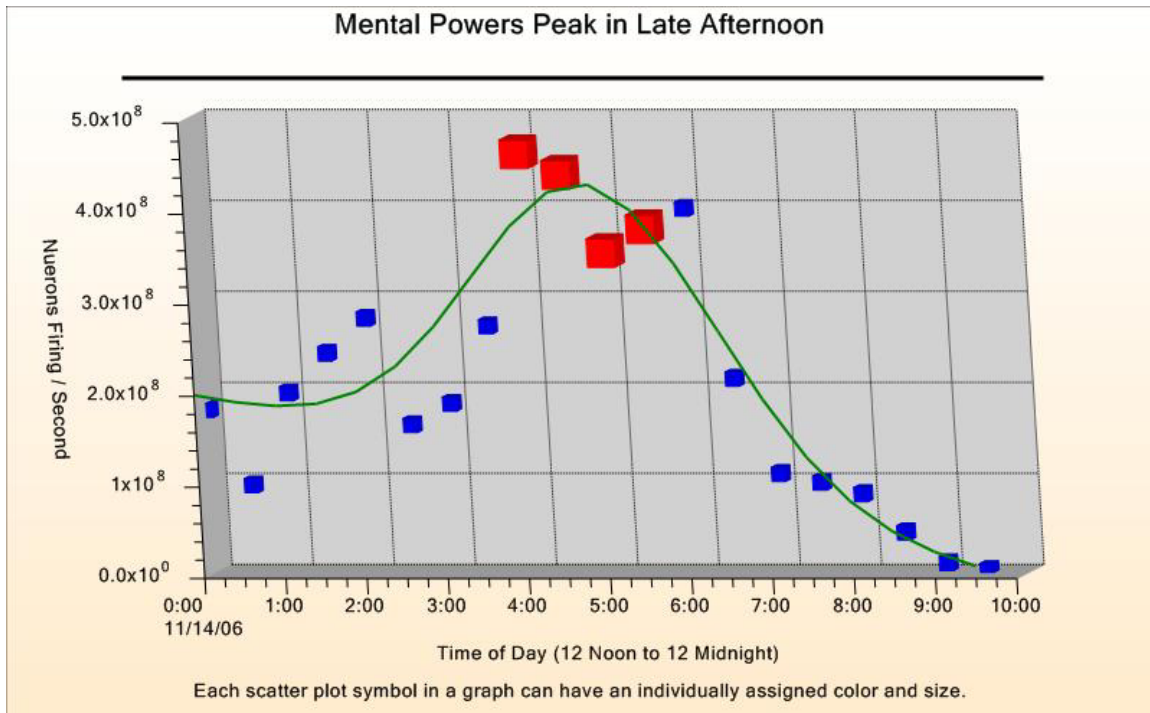
This class is a subclass of the **SimplePlot** class it displays simple datasets in a line plot format. Adjacent data points are connected using a straight, or a step line. The **SimpleRibbonPlot** class is a 2½ D chart, where the z-value of the plot is fixed using the associated datasets **ImplicitZValue** property. The depth of the ribbon is set using the associated datasets **ImplicitDepthValue** property.



3D line plots (`SimpleLinePlots.LinePlot3D`)

SimpleLinePlot

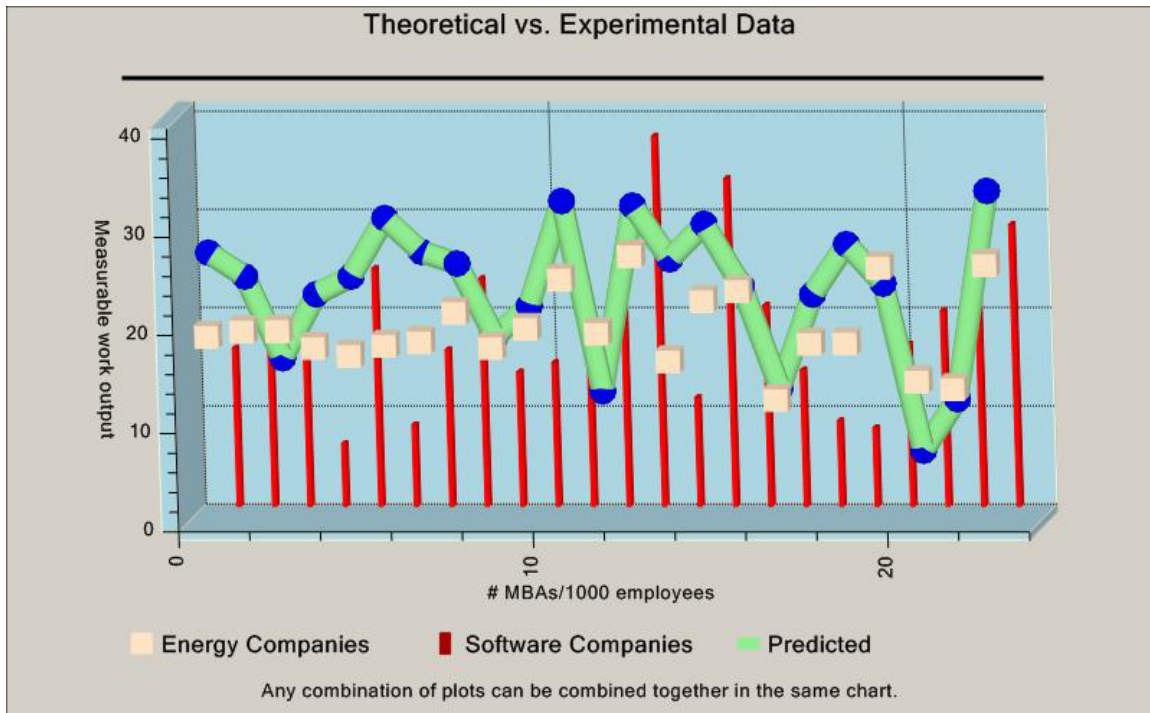
This class is a subclass of the **SimplePlot** class it displays simple datasets in a line plot format. Adjacent data points are connected using a straight, or a step line. The class will work with 2½ D data, using a constant z-value, set using the associated datasets **ImplicitZValue** property, or 3D data, where each data point is represented using individual x, y and z datapoints.



Scatter plots (`ScatterPlots.ScatterPoints`)

SimpleScatterPlot

This class is a subclass of the **SimplePlot** class and it displays simple datasets in a scatter plot format where each data point is represented using a symbol. The class will work with 2½ D data, using a constant z-value, set using the associated datasets **ImplicitZValue** property, or 3D data, where each data point is represented using individual x, y and z datapoints.



Pipeline plot (SimpleLinePlots.PipelineAndScatterPlot)

PipeLinePlot

This class is a subclass of the **SimplePlot** class and it displays simple datasets in a line plot format. Data points are represented using a sphere, and are connected using cylinder. The class will work with 2½ D data, using a constant z-value, set using the associated datasets **ImplicitZValue** property, or 3D data, where each data point is represented using individual x, y and z datapoints.

Legend Classes

LegendItem

BubblePlotLegendItem

Legend

StandardLegend

BubblePlotLegend

Legends provide a key for interpreting the various plot objects in a graph. It organizes a collection of legend items, one for each plot objects in the graph, and displays them in a rectangular frame.

Legend	This class is the abstract base class for chart legends.
LegendItem	This class is the legend item class for all plot objects except for bubble plots. Each legend item manages one symbol and descriptive text for that symbol. The StandardLegend class uses objects of this type as legend items.
BubblePlotLegendItem	This class is the legend item class for bubble plots. Each legend item manages a circle and descriptive text specifying the value of a bubble of this size. The BubblePlotLegend class uses objects of this type as legend items.
StandardLegend	This class is the legend class for all plot objects except for bubble plots. The legend item objects display in a row or column format. Each legend item contains a symbol and a descriptive string. The symbol normally associates the legend item to a particular plot object, and the descriptive string describes what the plot object represents.
BubblePlotLegend	This class is a legend class used exclusively with bubble plots. The legend item objects display as offset, concentric circles with descriptive text giving the key for the value associated with a bubble of this size.

Grid Classes

Grid

Grid lines are perpendicular to an axis, extending the major and/or minor tick marks of the axis across the width or height of the plot area volume of the chart.

Grid	This class defines the grid lines associated with an axis. Grid lines are perpendicular to an axis, extending the major and/or minor tick marks of the axis across the width or height of the plot area volume of the chart. This class works in conjunction with the LinearAxis , LogAxis and TimeAxis classes.

Chart Text Classes

ChartText

ChartTitle

AxisTitle

ChartLabel

NumericLabel

TimeLabel

StringLabel

The chart text classes draw one or more strings in the chart window. Different classes support different numeric formats, including floating point numbers, date/time values and multi-line text strings. International formats for floating point numbers and date/time values are also supported.

ChartText

This class draws a string in the current chart window. It is the base class for the **ChartTitle**, **AxisTitle** and **ChartLabel** classes. The **ChartText** class also creates independent text objects. Other classes that display text also use it internally.

ChartTitle

This class displays a text string as the title or footer of the chart.

AxisTitle

This class displays a text string as the title for an axis. The axis title position is outside of the axis label area. Axis titles for y-axes are rotated 90 degrees.

ChartLabel

This class is the abstract base class of labels that require special formatting.

NumericLabel

This class is a subclass of the **ChartLabel** class and it displays formatted numeric values.

TimeLabel

This class is a subclass of the **ChartLabel** class and it displays formatted **ChartCalendar** dates.

StringLabel

This class is a subclass of the **ChartLabel** class that formats string values for use as axis labels.

Miscellaneous Chart Classes

Marker
ChartImage
ChartShape
ChartSymbol
ChartSymbol3D

Various classes are used to position and draw objects that can be used as standalone objects in a graph, or as elements of other plot objects.

Marker	This class displays one of five marker types in a graph. The marker is used to create data cursors, or to mark data points. Markers will operate in 3D space.
ChartImage	This class encapsulates a System.Drawing.Image class, defining a rectangle in chart coordinates that the image is placed in. JPEG and other image files can be imported using the System.Drawing.Image class and displayed in a chart. Images can be place in 3D space, but the image itself will not be transformed and it will display as a 2D rectangular object with no rotation.
ChartShape	This class encapsulates a System.Drawing.Drawing2D.GraphicsPath class, placing the shape in a chart using a position defined in chart coordinates. A chart can display any object that can be defined using System.Drawing.Drawing2D.GraphicsPath class. Shapes can be place in 3D space, and should transform along with the rest of the chart.
ChartSymbol	This class defines 2D symbols used by the SimplePlot scatter plot functions. Pre-defined symbols: 2D Symbols: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE. 3D Symbols: HBAR3D, VBAR3D, CUBE3D, PLUS3D, DIAMOND3D, PYRAMID3D, HOURGLASS3D, ARROW3D, CYLINDER3D, SPHERE3D.
ChartSymbol3D	This class defines 3D symbols used by the SimplePlot scatter plot functions. Pre-defined 3D symbols include

cube, pyramid, diamond, plus, horizontal bar, vertical bar, 3D bar and cylinder, and sphere.

Mouse Interaction Classes

MouseListener

MoveObj

FindObj

DataToolTip

DataCursor

MoveData

ChartZoom

RotateButtonUserControl

RotateTranform3D

Several classes implement delegates for mouse events. The **MouseListener** class implements a generic interface for managing mouse events in a graph window. The **DataCursor**, **MoveData**, **MoveObj** and **ChartZoom** classes also implement mouse event delegates that use the mouse to mark, move and zoom chart objects and data.

MouseListener

This class implements .Net delegates that trap generic mouse events (button events and mouse motion events) that take place in a **ChartView** window. A programmer can derive a class from **MouseListener** and override the methods for mouse events, creating a custom version of the class.

MoveObj

This class extends the **MouseListener** class and it can select chart objects and move them. Moveable chart objects include axes, axes labels, titles, legends, arbitrary text, shapes and images. Use the **MoveData** class to move objects derived from **SimplePlot**.

FindObj

This class extends the **MouseListener** class, providing additional methods that selectively determine what graphical objects intersect the mouse cursor.

DataCursor

This class combines the **MouseListener** class and **Marker** class. Press a mouse button and the selected data cursor (horizontal and/or vertical line, cross hairs, or a small box) appears at the point of the mouse cursor. The data cursor tracks the mouse motion as long as the mouse button is pressed. Release the button and the data cursor disappears.

This makes it easier to line up the mouse position with the tick marks of an axis.

MoveData

This class selects and moves individual data points of an object derived from the **SimplePlot** class.

DataToolTip

A data tooltip is a popup box that displays the value of a data point in a chart. The data value can consist of the x-value, the y-value, x- and y-values, x-, y-, and x-values, group values and open-high-low-close values, for a given point in a chart.

CustomDataToolTip

A class that permits simple construction of custom data tooltips.

ChartZoom

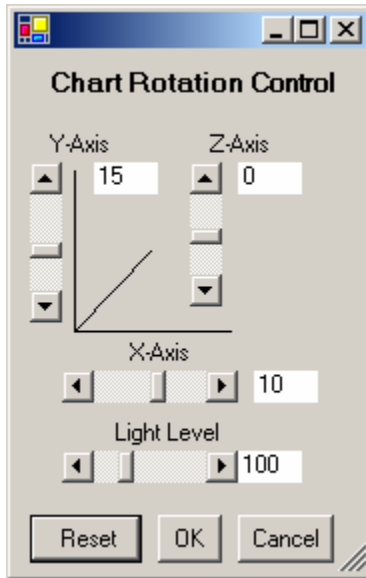
This class implements mouse controlled zooming for one or more simultaneous axes. The user starts zooming by holding down a mouse button with the mouse cursor in the plot area volume of a graph. The mouse is dragged and then released. The rectangle established by mouse start and stop points defines the new, zoomed, scale of the associated axes. Zooming has many different modes. Some of the combinations are:

One x or one y axis
 One x and one y axes
 One x and multiple y axes
 One y and multiple x axes
 Multiple x and y axes

RotateButtonUserControl

This mouse listener is a user control derived from a button

that looks like:  Clicking on it invokes the following dialog box:



that can rotate, and change the lighting level of one or more coordinate systems.

RotateTransform3D

This mouse listener monitors mouse clicks on chart objects type and invokes the dialog Chart Rotation Control dialog box.

File and Printer Rendering Classes

ChartPrint

BufferedImage

ChartPrint

This class implements printing using the .Net **System.Drawing.Printing** print-related services. It can select, setup, and output one or more ChartView objects to a printer.

BufferedImage

This class will convert a **ChartView** object to a .Net **Image** object. Optionally, the class saves the buffered image to an image file.

Miscellaneous Utility Classes

ChartCalendar

CSV

Dimension3D

GroupPoint3D
GroupPoint2D
DoubleArray
DoubleArray2D
BoolArray
Point3D
NearestPointData
TickMark
Polysurface
Rectangle2D
Rectangle3D

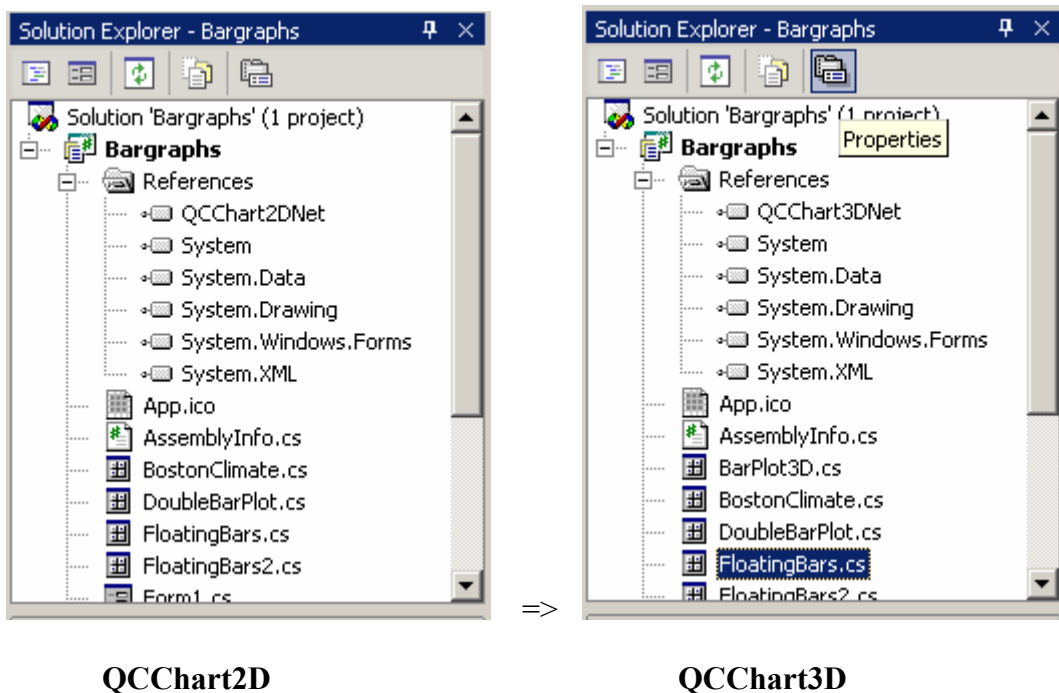
ChartCalendar	This class contains utility routines used to process ChartCalendar date objects.
CSV	This is a utility class for reading and writing CSV (Comma Separated Values) files.
Dimension3D	This is a utility class for handling dimension (height, width and depth) information using doubles, rather than the integers used by the Size class.
Point2D	Derived from Point3D and is included for compatibility with programs written for QCChart2D .
Point3D	This class encapsulates an xyx triplets of values as doubles (more useful in this software than the .Net Point and PointF classes).
GroupPoint2D	Derived from GroupPoint3D and is included for compatibility with programs written for QCChart2D .
GroupPoint3D	This class encapsulates an x-value, and an array of y-values, representing the x and y values of one column of a group data set, and a z-value.
DoubleArray	This class is used as an alternative to the standard .Net Array class, adding routines for resizing of the array, and the insertion and deletion of double based data elements.
DoubleArray2D	This class is used as an alternative to the standard .Net 2D Array class, adding routines for resizing of the array, and the insertion and deletion of double based data elements.

BoolArray	This class is used as an alternative to the standard .Net Array class, adding routines for resizing of the array, and the insertion and deletion of bool based data elements.
NearestPointData	This is a utility class for returning data that results from nearest point calculations.
TickMark	The axis classes use this class to to organize the location of the individual tick marks of an axis.
Polysurface	This is a utility class that defines complex 3D shapes as a list of simple 3-sided polygons. The contour plotting routines use it.
Rectangle2D	Derived from Rectangle3D and is included for compatibility with programs written for QCChart2D.
Rectangle3D	This is a utility class that specifies the x, y and z-value, along with the width, height and depth for a 3D rectangle.

Differences between the QCChart2D and QCChart3D API

Rather than invent a new API (API refers to the names, methods and properties in the software) we chose to use the API from the existing **QCChart2D**, adding new methods and creating new classes only when necessary. Because of this, programs written using **QCChart2D** are easily converted to **QCChart3D**. There are still few changes you must consistently make in your programs and these are summarized below.

1. The **QCChart3D** routines are in their own unique DLL (QCChart3DNet.DLL) and have their own unique namespace (com.quinncurtis.chart3dnet). This means that you should delete the QCChart2DNet.DLL from your program References node under the VS Solution Explorer, and add in its place the QCChart3DNet.DLL.



Next, change the namespace reference at the top any source files that use the charting classes from **com.quinncurtis.chart2dnet** to **com.quinncurtis.chart3dnet**.

```

using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
using com.quinncurtis.chart2dnet;

```

```

using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
using com.quinncurtis.chart3dnet;

```

QCChart2D

=>

QCChart3D

Several primitive classes that are specific to 2D charting must be changed to equivalent routines that contain z-dimension information. These are

QCChart2D => QCChart3D

Point2D Point3D

Dimension Dimension3D

Rectangle2D Rectangle3D

3. Since **QCChart2D** data would originally be stored in one of the dataset classes (**SimpleDataset**, **TimeSimpleDataset**, **GroupDataset**, **TimeGroupDataset**), they would not contain explicit z-dimension information. The **QCChart3D** versions of these classes include two important properties that you use to define the z-dimension position and depth of 3D objects like bars, ribbons and scatter plot symbols. These would be the **ChartDataset.ImplicitZValue** and **ChartDataset.ImplicitDepthValue** properties.

```

TimeGroupDataset Dataset1 =
    new TimeGroupDataset("Stock Data",xValues,stockPriceData);
Dataset1.ImplicitZValue = 0.5;
Dataset1.ImplicitDepthValue = 0.5;

```

4. The default rotational state of any newly created coordinate system (**CartesianCoordinates** or **TimeCoordinates**) is un-rotated (0 degrees) about all three axes. Give the coordinate system an initial rotational state using the coordinate systems **AbsRotateCoordinateSystem** method. The example below rotates the chart 10 degrees about the x-axis and 20 degrees about the y-axis.

```

pTransform1 = new TimeCoordinates();

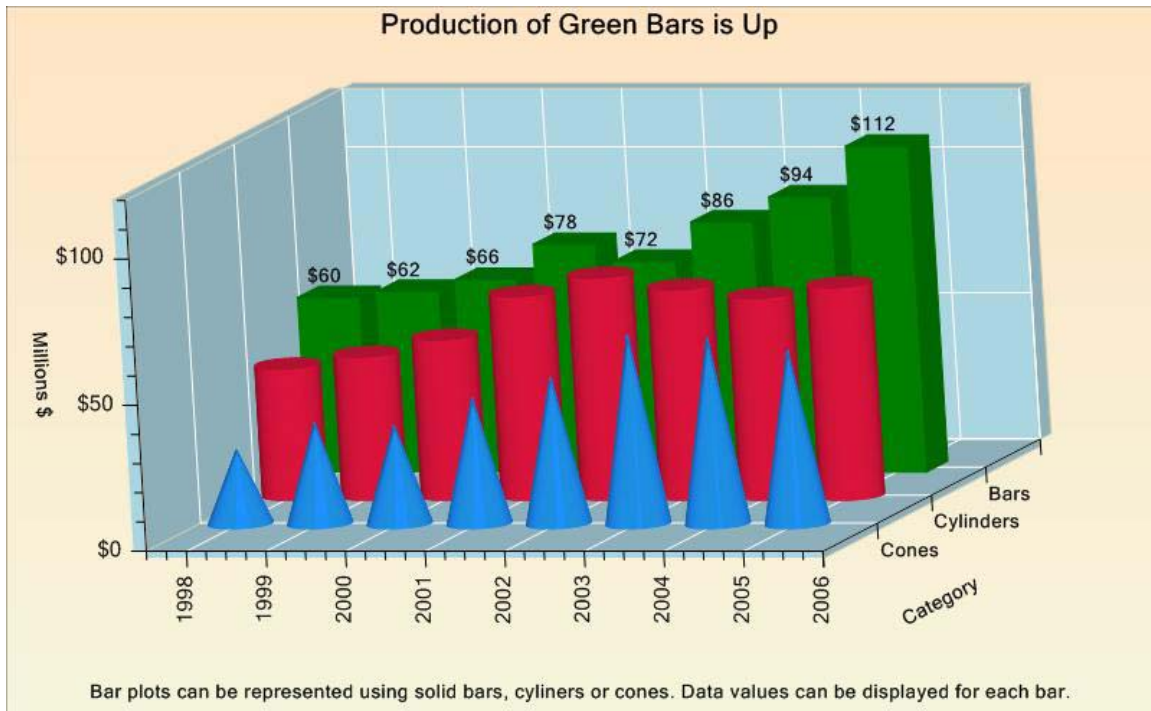
```

```

.
.
.
// Set rotational parameters
Point3D protate = new Point3D(10, 20, 0);
pTransform1.AbsRotateCoordinateSystem( protate);

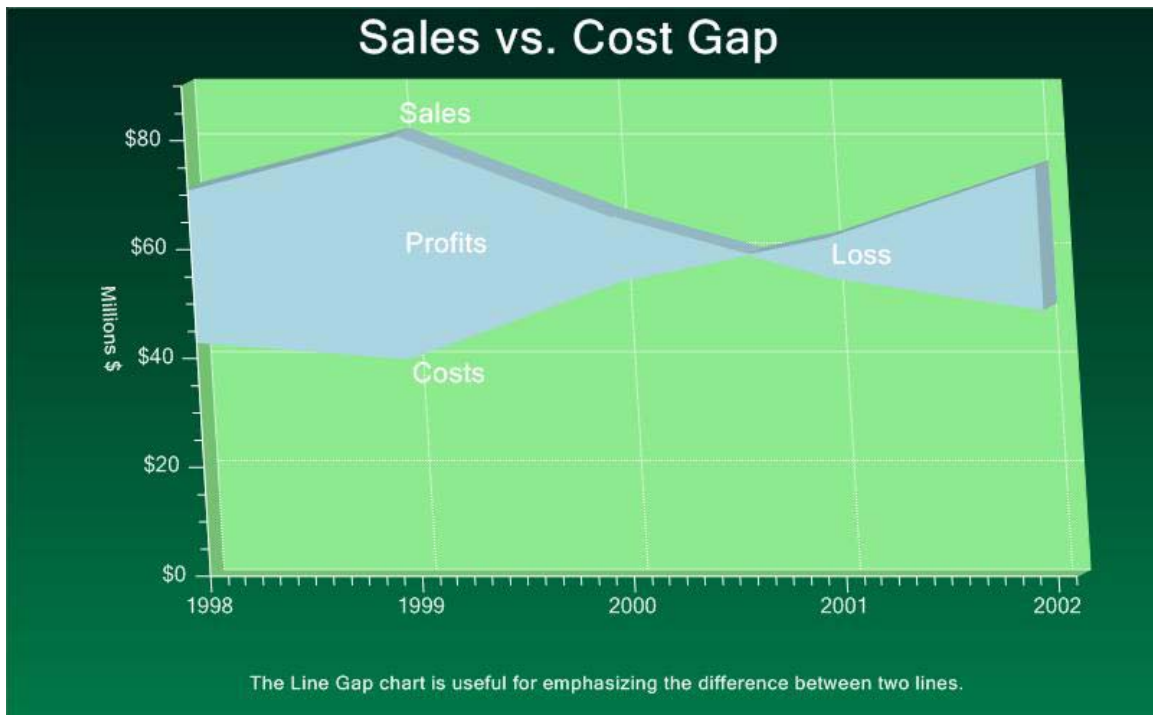
```

5. The relative depth of the z-dimension is set using the **ChartView** method, **SetFractionalZViewportDepth**. It sets the depth of the z-dimension as a fraction of the **ChartView** width. The default value of the z-dimension depth is 0.3, meaning that the apparent depth of the x-dimension will be 30% of the apparent width of the **ChartView**. Making the value larger will give the 3D chart more depth, smaller values mean less depth. The two examples below show how changes in the value can affect the appearance of a chart.



(Bargraphs.SimpleBars)

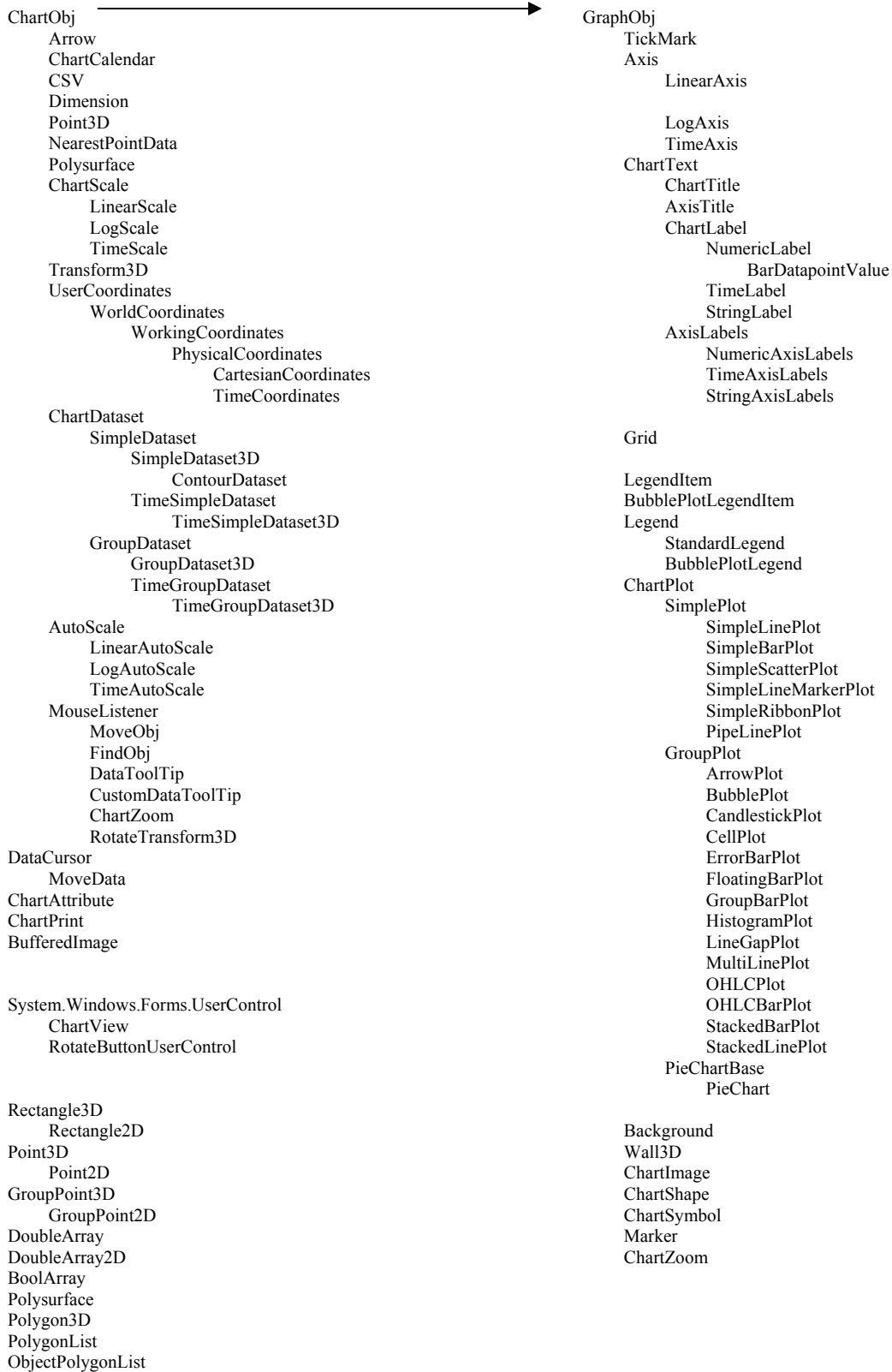
```
chartVu.SetFractionalZViewportDepth(0.8);
```



(MiscCharts.LineGap)

```
chartVu.SetFractionalZViewportDepth(0.05);
```


Class Hierarchy of QCChart3D



3. Chart Datasets

ChartDataset

SimpleDataset

SimpleDataset3D

ContourDataset

TimeSimpleDataset

TimeSimpleDataset3D

GroupDataset

GroupDataset3D

TimeGroupDataset

TimeGroupDataset3D

The dataset classes organize the numeric data associated with a plot object. Plot objects are chart objects derived from the **ChartPlot** class. There are two major types of data supported by the dataset classes. The first is simple xy data, where for every x-value there is one y-value. The second data type is group data, where every x-value can have one or more y-values. A couple of variants of the simple xy datasets include a simple dataset type that can substitute **ChartCalendar** values as the x-values, and a dataset type that is used to plot contour data. A variant of the group dataset type also uses **ChartCalendar** values as the x-values.

Copies of the original data arrays are stored. The original source data can be deleted once the dataset is created. If you want to make any changes to the data, you must change the data in the dataset, not the original source data.

Datasets can be initialized using CSV (comma separated value) files. The CSV file is a common file structure that can share data between spreadsheets, databases and word processing programs. Datasets can also write CSV files, loadable into other programs.

If you need to plot data stored in a database, either save the data as a CSV file, or read the data into arrays. Once the data is in either format, initialize a dataset using the appropriate class and constructor.

The **ChartDataset** class is the abstract base class for all of the dataset classes. It contains data common to all dataset classes, such as the x-value array, the number of x-values, the dataset name and the dataset type.

2½ D Datasets and 3D Datasets

The dataset classes that do NOT end in 3D (**SimpleDataset**, **TimeSimpleDataset**, **GroupDataset**, and **TimeGroupDataset**) represent 2½ D data. The concept of 2½ D data is easy to understand if you think of the way most 3D charts are used. Data which is normally 2D, i.e. a set of x , y data points, is displayed in three dimensions using 3D bars,

ribbons, 3D symbols, or some other 3D solid. The 2D data is displayed along a plane of constant z-value, which is fixed for a given dataset, but can vary from dataset to dataset. Giving the plots associated with the datasets different z-values will give the plot a layered look. The z-value associated with a 2½ D dataset is set using the datasets **ImplicitZValue** property. There is a related property, **ImplicitDepthValue**, that defines the depth of plot objects (those that look to a depth parameter), along the z-axis. By default, the z-axis of a 3D transform is scaled from 0-1, with 0 representing the screen, and 1 representing the maximum displayable depth into the screen. The default **ImplicitZValue** is set to 0.0, and the default **ImplicitDepthValue** is 0.1.

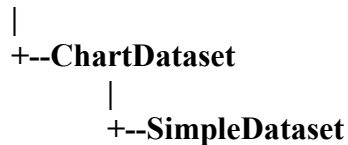
The 3D datasets end in 3D ((**SimpleDataset3D**, **TimeSimpleDataset3D**, **GroupDataset3D**, and **TimeGroupDataset3D**). Instead of having a fixed z-value for all xy data values in the dataset, a 3D dataset has an additional array that stores a unique z-data value for each xy data value in the dataset. When a plot object is connected to 3D dataset, the z-value of the plot can vary point to point.

The **ContourDataset** is the exception that proves the rule. It does not end in 3D, but it is a 3D dataset type. We did not add the 3D suffix to maintain compatibility with the **QCChart2D** API.

Simple Numeric Dataset

Class SimpleDataset

ChartObj



The **SimpleDataset** class represents simple floating point xy data, where for every x-value there is one y-value. The number of xy data points in a simple dataset is referred to as the number of columns, or as the property **numberDatapoints**. Think of a spreadsheet file that looks like:

x-values	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
y-values	y[0]	y[1]	y[2]	y[3]	y[4]	y[5]
ImplicitZValue	0					

number of xy data pairs = numberDatapoints = numberColumns = 6

This would be the ROW_MAJOR format if the data were stored in a CSV file.

It has two main constructors. This constructor creates a dataset using the x-, y-values stored in arrays.

SimpleDataset constructors

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As Double(), _
    ByVal y As Double() _
)

Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As Double(), _
    ByVal y As Double(), _
    ByVal zvalue As Double, _
    ByVal depthvalue As Double _
)

[C#]
public SimpleDataset(
    string sname,
    double[] x,
    double[] y
);

public SimpleDataset(
    string sname,
    double[] x,
    double[] y,
    double zvalue,
    double depthvalue
);
```

<i>sname</i>	Specifies the name of the dataset.
<i>x</i>	An array that specifies the x-values of a dataset.
<i>y</i>	An array that specifies the y-values of a dataset. The length of the y array must match the length of the x array.
<i>zvalue</i>	Sets the ImplicitZValue for the dataset. The default value is 0.0;
<i>depthvalue</i>	Sets the ImplicitDepthValue for the dataset. The default value is 0.1.

The number of data points is the value of `x.Length` property. The x and y arrays must be the same length and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

The next constructor creates a dataset using the x- and y-values stored in a file that uses the CSV (Comma Separated Value) format. There are two ways to organize the numeric values in the data file. If you use the `COLUMN_MAJOR` format, the first column

represents the x-values and the second column the y-values. If you use the ROW_MAJOR format, the first row represents the x-values and the second row the y-values. Use the **CSV.SetOrientation** method to initialize the csv argument for the proper data orientation.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal csv As CSV, _
    ByVal filename As String, _
    ByVal rowskip As Integer, _
    ByVal columnskip As Integer _
)
```

```
[C#]
public SimpleDataset(
    CSV csv,
    string filename,
    int rowskip,
    int columnskip
);
```

<i>csv</i>	An instance of a CSV object.
<i>filename</i>	The name of the file.
<i>rowskip</i>	Skip this many rows before starting the read operation.
<i>columnskip</i>	For each row of data, skip this many columns before reading the first value from the row.

You can retrieve references to the internal arrays used to store the data using the **SimpleDataset** methods **GetXData** and **GetYData**. Change the values in the data using these references. You can also modify a point at a time using one of the **SetDataPoint** methods. If you need to add new points to a dataset, increasing its size, use one of the **AddDataPoint**, or **InsertDataPoint** methods. Delete data points using the **DeleteDataPoint** method. In order to see the modified dataset, force the graph to redraw using **ChartView.UpdateDraw** method. The indexed accessor property of the **SimpleGroupDataset** will get or set a datapoint as a **Point3D** object.

Example of creating simple datasets from numeric arrays

[Visual Basic]

```
Dim x1() As Double = {10, 20, 30, 40, 50}
```

```

Dim y1() As Double = {9, -21, 20, 40, 30}
Dim zvalue As Double = 0.5
Dim depthvalue As Double 0.4
Dim Dataset1 As SimpleDataset = _
    new SimpleDataset("First", x1, y1, zvalue, depthvalue)

Dim n2 As Integer = 9
Dim x2(n2 - 1) As Double ' Dim'd dimension is upper limit, not size
Dim y2(n2 - 1) As Double ' Dim'd dimension is upper limit, not size

x2(0) = 5
x2(1) = 7
.
.
x2(n2 - 1) = 100

y2(0) = 15
y2(1) = 25
.
.
y2(n2 - 1) = 100
zvalue = 0.0
Dim Dataset2 As SimpleDataset = _
    New SimpleDataset("Second", x2, y2 y2, zvalue, depthvalue)

```

[C#]

```

double [] x1 = {10, 20, 30, 40, 50};
double [] y1 = {9, -21, 20, 40, 30};
double zvalue = 0.5;
double depthvalue = 0.4;
SimpleDataset Dataset1 = new SimpleDataset("First", x1, y1, zvalue, depthvalue);

int n2 = 9;
double []x2 = new double[n2]; // dimension is size, not upper limit
double []y2 = new double[n2]; // dimension is size, not upper limit

x2[0] = 5;
x2[1] = 7;
//.
//.
x2[n2 - 1] = 100;

```

```

y2[0] = 15;
y2[1] = 25;
//.
//.
y2[n2 - 1] = 100;
zvalue = 0.0;
SimpleDataset Dataset2 = new SimpleDataset("Second", x2, y2, zvalue, depthvalue);

```

Example of reading and writing a simple dataset from a CSV file

[C#]

```

CSV csvdata = new CSV();
SimpleDataset Dataset1 =
    new SimpleDataset(csvdata, "SimpleDataset.csv", 0, 0);
    // Write out dataset as a CSV file under a different file name
Dataset1.ImplicitZValue = 0.5;
Dataset1.ImplicitDepthValue = 0.5;
Dataset1.WriteSimpleDataset(csvdata, "SimpleDataset2.csv");

```

[Visual Basic]

```

Dim csvdata As CSV = New CSV()
Dim Dataset1 As SimpleDataset =
    New SimpleDataset(csvdata, "SimpleDataset.csv", 0, 0)
Dataset1.ImplicitZValue = 0.5
Dataset1.ImplicitDepthValue = 0.5

' Write out dataset as a CSV file under a different file name
Dataset1.WriteSimpleDataset(csvdata, "SimpleDataset2.csv")

```

Example of modifying simple dataset elements using the indexed accessor property.

[C#]

```

// Define a simple dataset
SimpleDataset Dataset1 = new SimpleDataset("First", x1, y1);
Point3D datapoint = Dataset1[0]; // Get the xy point at index 0 in the dataset

```



```

if datapoint.X < 0 datapoint.X = Math.Abs(datapoint.X); // arbitrary
Dataset1[0] = datapoint; // Change the datapoint

```

[Visual Basic]

```

Dim Dataset1 As SimpleDataset = New SimpleDataset("First", x1, y1)
Dim datapoint As Point3D = Dataset1(0) 'Get the xy point at index 0 in the dataset
If datapoint.X < 0 Then datapoint.X = Math.Abs(datapoint.X) ' arbitrary
Dataset1(0) = datapoint ' Change the datapoint

```

Example of modifying the implicit z-value and implicit depth value using dataset property.

[C#]

```

// Define a simple dataset , used default z-value and depth value
SimpleDataset Dataset1 = new SimpleDataset("First",x1,y1); //
Dataset1.ImplicitZValue = 0.3;
Dataset1.ImplicitDepthValue = 0.5;

```

[Visual Basic]

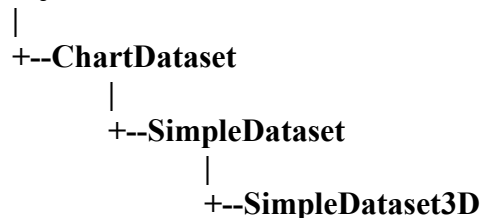
```

` Define a simple dataset , used default z-value and depth value
Dim Dataset1 As SimpleDataset = New SimpleDataset("First", x1, y1)
Dataset1.ImplicitZValue = 0.3
Dataset1.ImplicitDepthValue = 0.5

```

Class SimpleDataset3D

ChartObj



The **SimpleDataset3D** class represents a collection of 3D data points, where each data point can have a unique x, y and z- value. The number of xy data points in a simple dataset is referred to as the number of columns, or as the property **numberDatapoints**. Think of a spreadsheet file that looks like:

70 Chart Datasets

x-values	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
y-values	y[0]	y[1]	y[2]	y[3]	y[4]	y[5]
z-values	z[0]	z[1]	z[2]	z[3]	z[4]	z[5]

number of xyz data points = numberDatapoints = numberColumns = 6

This would be the ROW_MAJOR format if the data were stored in a CSV file.

It has two main constructors. This constructor creates a dataset using the x-, y-values stored in arrays.

SimpleDataset3D constructors

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As Double(), _
    ByVal y As Double() _
    ByVal z As Double() _
)

Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As Double(), _
    ByVal y As Double(), _
    ByVal z As Double(), _
    ByVal depthvalue As Double _
)

[C#]
public SimpleDataset3D(
    string sname,
    double[] x,
    double[] y,
    double[] z,
);

public SimpleDataset3D(
    string sname,
    double[] x,
    double[] y,
    double[] z,
    double depthvalue
);
```

sname Specifies the name of the dataset.

x An array that specifies the x-values of a dataset.

<i>y</i>	An array that specifies the y-values of a dataset. The length of the y array must match the length of the x array.
<i>z</i>	An array that specifies the z-values of a dataset. The length of the z array must match the length of the x array.
<i>depthvalue</i>	Sets the ImplicitDepthValue for the dataset. The default value is 0.1.

The number of data points is the value of `x.Length` property. The x, y and z arrays must be the same length and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

The next constructor creates a dataset using the x-, y- and z-values stored in a file that uses the CSV (Comma Separated Value) format. There are two ways to organize the numeric values in the data file. If you use the `COLUMN_MAJOR` format, the first column represents the x-values, the second column the y-values and the third column the z-values. If you use the `ROW_MAJOR` format, the first row represents the x-values, the second row the y-values and the third row the z-values. Use the **CSV.SetOrientation** method to initialize the csv argument for the proper data orientation.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal csv As CSV, _
    ByVal filename As String, _
    ByVal rowskip As Integer, _
    ByVal columnskip As Integer _
)
```

```
[C#]
public SimpleDataset3D(
    CSV csv,
    string filename,
    int rowskip,
    int columnskip
);
```

<i>csv</i>	An instance of a CSV object.
<i>filename</i>	The name of the file.
<i>rowskip</i>	Skip this many rows before starting the read operation.
<i>columnskip</i>	For each row of data, skip this many columns before reading the first value from the row.

You can retrieve references to the internal arrays used to store the data using the **SimpleDataset3D** methods **GetXData**, **GetYData** and **GetZData**. Change the values in the data using these references. You can also modify a point at a time using one of the **SetDataPoint** methods. If you need to add new points to a dataset, increasing its size, use one of the **AddDataPoint**, or **InsertDataPoint** methods. Delete data points using the **DeleteDataPoint** method. In order to see the modified dataset, force the graph to redraw using **ChartView.UpdateDraw** method. The indexed accessor property of the **SimpleGroupDataset** will get or set a datapoint as a **Point3D** object.

Example of creating simple datasets from numeric arrays

[Visual Basic]

```
Dim x1() As Double = {10, 20, 30, 40, 50}
Dim y1() As Double = {9, -21, 20, 40, 30}
Dim z1() As Double = {0.5, 0.5, 0.5, 0.5, 0.5}
Dim depthvalue As Double 0.4
Dim Dataset1 As new SimpleDataset3D("First", x1, y1, z1, depthvalue)
```

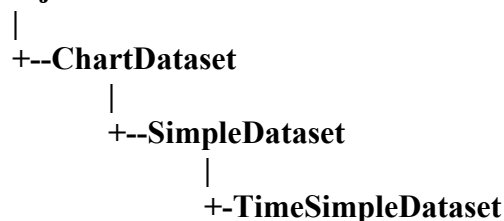
[C#]

```
double [] x1 = {10, 20, 30, 40, 50};
double [] y1 = {9, -21, 20, 40, 30};
double [] z1 = {0.5, 0.5, 0.5, 0.5, 0.5};
double depthvalue = 0.4;
SimpleDataset3D Dataset1 = new SimpleDataset3D("First", x1, y1, z1, depthvalue);
```

Simple Date/Time Dataset

Class TimeSimpleDataset

ChartObj



The **TimeSimpleDataset** uses **ChartCalendar** dates as the x-values, and floating point numbers as the y-values. **ChartCalendar** values are actually stored internally as their equivalent millisecond values. The **TimeSimpleDataset** class adds a large number of methods to the **SimpleDataset** class that make it easy to create and modify datasets that use **ChartCalendar** values.

It has two main constructors. The following constructor creates a time dataset using the x- and y-values stored in arrays.

TimeSimpleDataset constructors

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As ChartCalendar(), _
    ByVal y As Double() _
)

Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As ChartCalendar(), _
    ByVal y As Double(), _
    ByVal zvalue As Double, _
    ByVal depthvalue As Double _
)

[CS#]
public TimeSimpleDataset(
    string sname,
    ChartCalendar[] x,
    double[] y,
    double zvalue,
    double depthvalue
);

public TimeSimpleDataset(
    string sname,
    ChartCalendar[] x,
    double[] y
);
```

<i>sname</i>	Specifies the name of the dataset.
<i>x</i>	An array that specifies the ChartCalendar x-values of a dataset.
<i>y</i>	An array that specifies the y-values of a dataset. The length of the y array must match the length of the x array.
<i>zvalue</i>	Sets the ImplicitZValue for the dataset. The default value is 0.0;

depthvalue Sets the **ImplicitDepthValue** for the dataset. The default value is 0.1.

The number of data points is the value of `x.Length` property. The `x` and `y` arrays must be the same length and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

The next constructor creates a time dataset using the `x`- and `y`-values stored in a file that uses the CSV (Comma Separated Value) format. There are two ways to organize the numeric values in the data file. If you use the `COLUMN_MAJOR` format, the first column represents the time values and the second column the `y`-values. If you use the `ROW_MAJOR` format, the first row represents the time values and the second row the `y`-values. Use the **CSV.SetOrientation** method to initialize the `csv` argument for the proper data orientation.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal csv As CSV, _
    ByVal filename As String, _
    ByVal rowskip As Integer, _
    ByVal columnskip As Integer _
)

[C#]
public TimeSimpleDataset(
    CSV csv,
    string filename,
    int rowskip,
    int columnskip
);
```

csv An instance of a **CSV** object.

filename The name of the file.

rowskip Skip this many rows before starting the read operation.

columnskip For each row of data, skip this many columns before starting this read operation.

A **DateTimeFormatInfo** object, and a date time format string, in the **CSV** class, control the interpretation of the **ChartCalendar** values. The format in the file must match the format specified for the **CSV** class. The underlying conversion mechanism calls the **DateTime.ToString(String formatstring, DateTimeFormatInfo info)** method for the conversion. The default format for the date time *formatstring* object is "M/dd/yy". Call

the **SetDateTimeFormatString** method to change the default date time format. See the documentation for the .Net **DateTime.ToString** method to figure out the various formatting options for the date time format string. If you are into internationalization (and difficult to understand .Net documentation), you can also create your own **DateTimeFormatInfo** object, installing it in the **CSV** object using **CSV.SetTimeDateFormat** method. The date time format string and the **DateTimeFormatInfo** object apply to both CSV files used for input, and CSV files used for output. If an attempt is made to read date/time values that do not match the desired format, the data values are set to invalid date/time values.

You can retrieve a *copy* of the date time data using the **TimeSimpleDataset.GetTimeXData** method. It returns an array of **ChartCalendar** objects, and it is *not* a reference to the underlying data. The underlying data is stored as double values that represent the millisecond equivalent of the date time values. The **TimeSimpleDataset** **GetXData** and **GetYData** methods return references to the underlying data. You can also modify a point at a time using one of the **SetTimeDataPoint**, **SetTimeXDataValue** and **SetYDataValue** methods. If you need to add new points to the dataset, increasing its size, use one of the **AddTimeDataPoint**, or **InsertTimeDataPoint** methods. Delete data points using the **DeleteTimeDataPoint** method. In order to see the modified dataset, force the graph to redraw using **ChartView.UpdateDraw** method.

Example of creating a simple time datasets

[C#]

```
int numpnts = 32;
ChartCalendar [] x1= new ChartCalendar[numpnts];
double []y1 = new double[numpnts];
double []y2 = new double[numpnts];
y1[0] = 100;
y2[0] = 30;
x1[0] = (ChartCalendar) currentdate.Clone();
currentdate.Add(ChartObj.MONTH,3);
for (i=1; i < numpnts; i++)
{
    x1[i] = (ChartCalendar) currentdate.Clone();
    y1[i] += y1[i-1] + (5 + i) * (0.75 - ChartSupport.GetRandomDouble());
    y2[i] += y2[i-1] + (15 + i) * (0.95 - ChartSupport.GetRandomDouble());
    currentdate.Add(ChartObj.MONTH,3);
}
// Use default z-value and depth value
TimeSimpleDataset Dataset1 = new TimeSimpleDataset("Sales",x1,y1);
```

76 Chart Datasets

```
double zvalue = 0.1;
double depthvalue = 0.4;
TimeSimpleDataset Dataset2 =
    new TimeSimpleDataset("Expenses",x1,y2, zvalue, depthvalue);
```

[Visual Basic]

```
Dim numpnts As Integer = 32
Dim x1(numpnts-1) As ChartCalendar
Dim y1(numpnts-1) As Double
Dim y2(numpnts-1) As Double
Dim currentdate As New ChartCalendar(1998, ChartObj.JANUARY, 1)
y1(0) = 100
y2(0) = 30
x1(0) = currentdate.Clone()
currentdate.Add(ChartObj.MONTH, 3)
For i = 1 To numpnts - 1
    x1(i) = currentdate.Clone()
    y1(i) += y1((i - 1)) + (5 + i) * (0.75 - ChartSupport.GetRandomDouble())
    y2(i) += y2((i - 1)) + (15 + i) * (0.95 - ChartSupport.GetRandomDouble())
    currentdate.Add(ChartObj.MONTH, 3)
Next i
' Use default z-value and depth value
Dim Dataset1 As New TimeSimpleDataset("Sales", x1, y1)
Dim zvalue As Double = 0.1
Dim depthvalue As Double = 0.4
Dim Dataset2 As New TimeSimpleDataset("Expenses", x1, y2, zvalue, depthvalue)
```

Example of creating a simple time datasets from a CSV file

[C#]

```
// Default time date format is "M/dd/yyyy"
CSV csvDataFile = new CSV();

// Create a dataset based on a previously saved csv file
TimeSimpleDataset Dataset1 =
    new TimeSimpleDataset(csvDataFile, " LineFill.Dataset1.csv ",0,0);
Dataset1.ImplicitZValue = 0.5;
Dataset1.ImplicitDepthValue = 0.5;

// Write out dataset as a CVS file
```



```

Dataset1.WriteTimeSimpleDataset(csv, "LineFill.Dataset1.csv");

// Read it back in just as a test
Dataset1.ReadTimeSimpleDataset(csv, "LineFill.Dataset1.csv", 0, 0);

```

[Visual Basic]

```

'Default time date format is "M/dd/yyyy"
Dim csvDataFile As CSV = New CSV()

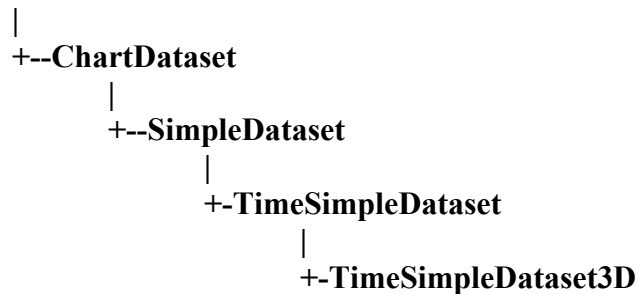
'Create a dataset based on a previously saved csv file
Dim Dataset1 As TimeSimpleDataset = _
    New TimeSimpleDataset(csvDataFile, " LineFill.Dataset1.csv ", 0, 0)
Dataset1.ImplicitZValue = 0.5
Dataset1.ImplicitDepthValue = 0.5

'Write out dataset as a CVS file
Dataset1.WriteTimeSimpleDataset(csvDataFile, "LineFill.Dataset1.csv")
' Read it back in just as a test
Dataset1.ReadTimeSimpleDataset(csvDataFile, "LineFill.Dataset1.csv", 0, 0)

```

Class TimeSimpleDataset3D

ChartObj



The **TimeSimpleDataset3D** class represents true 3D data, where every data point can have a unique x, y and z data value. It uses **ChartCalendar** dates as the x-values, and floating point numbers as the y- and z-values. **ChartCalendar** values are actually stored internally as their equivalent millisecond values. The **TimeSimpleDataset** class adds a large number of methods to the **SimpleDataset** class that make it easy to create and modify datasets that use **ChartCalendar** values.

It has two main constructors. The following constructor creates a time dataset using the x- and y-values stored in arrays.

TimeSimpleDataset3D constructors

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As ChartCalendar(), _
    ByVal y As Double() _
    ByVal z As Double() _
)

Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As ChartCalendar(), _
    ByVal y As Double(), _
    ByVal z As Double() _
    ByVal depthvalue As Double _
)

[C#]
public TimeSimpleDataset3D(
    string sname,
    ChartCalendar[] x,
    double[] y,
    double[] z,
    double depthvalue
);

public TimeSimpleDataset3D(
    string sname,
    ChartCalendar[] x,
    double[] y,
    double[] z
);

```

<i>sname</i>	Specifies the name of the dataset.
<i>x</i>	An array that specifies the ChartCalendar x-values of a dataset.
<i>y</i>	An array that specifies the y-values of a dataset. The length of the y array must match the length of the x array.
<i>z</i>	An array that specifies the z-values of a dataset. The length of the z array must match the length of the x array.
<i>depthvalue</i>	Sets the ImplicitDepthValue for the dataset. The default value is 0.1.

The number of data points is the value of `x.Length` property. The x and y arrays must be the same length and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

The next constructor creates a time dataset using the x- and y-values stored in a file that uses the CSV (Comma Separated Value) format. There are two ways to organize the numeric values in the data file. If you use the COLUMN_MAJOR format, the first column represents the time values and the second column the y-values. If you use the ROW_MAJOR format, the first row represents the time values and the second row the y-values. Use the **CSV.SetOrientation** method to initialize the csv argument for the proper data orientation.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal csv As CSV, _
    ByVal filename As String, _
    ByVal rowskip As Integer, _
    ByVal columnskip As Integer _
)

[C#]
public TimeSimpleDataset3D(
    CSV csv,
    string filename,
    int rowskip,
    int columnskip
);
```

<i>csv</i>	An instance of a CSV object.
<i>filename</i>	The name of the file.
<i>rowskip</i>	Skip this many rows before starting the read operation.
<i>columnskip</i>	For each row of data, skip this many columns before starting this read operation.

A **DateTimeFormatInfo** object, and a date time format string, in the **CSV** class, control the interpretation of the **ChartCalendar** values. The format in the file must match the format specified for the **CSV** class. The underlying conversion mechanism calls the **DateTime.ToString(String formatstring, DateTimeFormatInfo info)** method for the conversion. The default format for the date time *formatstring* object is "M/dd/yy". Call the **SetDateTimeFormatString** method to change the default date time format. See the documentation for the .Net **DateTime.ToString** method to figure out the various formatting options for the date time format string. If you are into internationalization (and difficult to understand .Net documentation), you can also create your own **DateTimeFormatInfo** object, installing it in the **CSV** object using **CSV.SetTimeDateFormat** method. The date time format string and the **DateTimeFormatInfo** object apply to both CSV files used for input, and CSV files used for output. If an attempt is made to read date/time values that do not match the desired format, the data values are set to invalid date/time values.

You can retrieve a *copy* of the date time data using the **TimeSimpleDataset.GetTimeXData** method. It returns an array of **ChartCalendar** objects, and it is *not* a reference to the underlying data. The underlying data is stored as double values that represent the millisecond equivalent of the date time values. The **TimeSimpleDataset** **GetXData** and **GetYData** methods return references to the underlying data. You can also modify a point at a time using one of the **SetTimeDataPoint**, **SetTimeXDataValue** and **SetYDataValue** methods. If you need to add new points to the dataset, increasing its size, use one of the **AddTimeDataPoint**, or **InsertTimeDataPoint** methods. Delete data points using the **DeleteTimeDataPoint** method. In order to see the modified dataset, force the graph to redraw using **ChartView.UpdateDraw** method.

.

Example of creating a simple time datasets

[C#]

```
int numpnts = 32;
ChartCalendar currentdate = new ChartCalendar();
ChartCalendar [] x1= new ChartCalendar[numpnts];
double []y1 = new double[numpnts];
double []z1 = new double[numpnts];
y1[0] = 100;
z1[0] = 0.1;
x1[0] = (ChartCalendar) currentdate.Clone();
currentdate.Add(ChartObj.MONTH,3);
for (i=1; i < numpnts; i++)
{
    x1[i] = (ChartCalendar) currentdate.Clone();
    y1[i] += y1[i-1] + (5 + i) * (0.75 - ChartSupport.GetRandomDouble());
    z1[i] = 0.1 + i * 0.02;
    currentdate.Add(ChartObj.MONTH,3);
}
TimeSimpleDataset3D Dataset1 = new TimeSimpleDataset3D("Sales",x1,y1, z1);
Dataset1.ImplicitDepthValue = 0.4;
```

[Visual Basic]

```
Dim currentdate As New ChartCalendar()
Dim numpnts As Integer = 32
Dim x1(numpnts - 1) As ChartCalendar
Dim y1(numpnts - 1) As Double
Dim z1(numpnts - 1) As Double
```

```

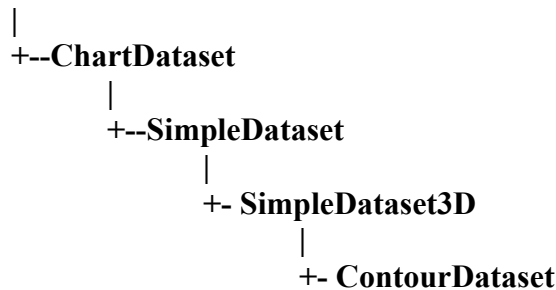
y1(0) = 100
z1(0) = 0.1
x1(0) = CType(currentdate.Clone(), ChartCalendar)
currentdate.Add(ChartObj.MONTH, 3)
For i = 1 To numpnts - 1
    x1(i) = CType(currentdate.Clone(), ChartCalendar)
    y1(i) += y1((i - 1)) + (5 + i) * (0.75 - ChartSupport.GetRandomDouble())
    z1(i) = 0.1 + i * 0.02
    currentdate.Add(ChartObj.MONTH, 3)
Next i
Dim Dataset1 As New TimeSimpleDataset3D("Sales", x1, y1, z1)
Dataset1.ImplicitDepthValue = 0.4

```

ContourDataset Dataset

Class ContourDataset

ChartObj



The **ContourDataset** adds a third dimension (z-values) to the x- and y- values of the simple dataset. Use it with the **SimplePlot**, **GroupPlot** and the **ContourPlot** class.

ContourDataset constructors

This constructor creates a new **ContourDataset** object that represents a surface formed by a regular grid in the xy plane. The number of objects in the **Point3D** array must equal (rows * columns) and must form an even grid in the xy plane.

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal grid As Point3D(), _
    ByVal rows As Integer, _
    ByVal columns As Integer _
)

[C#]
public ContourDataset(
    string sname,
    Point3D[] grid,

```

```

    int rows,
    int columns
);

```

This constructor creates a new **ContourDataset** object that represents a surface, not necessarily a regular grid. A triangularization algorithm calculates the interconnection of the vertices defining the surface.

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal grid As Point3D() _
)

[C#]
public ContourDataset(
    string sname,
    Point3D[] grid
);

```

This constructor creates a new **ContourDataset** object that represents a surface, not necessarily a regular grid. A triangularization algorithm calculates the interconnection of the vertices defining the surface. The length of the x, y and z arrays must match.

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As Double(), _
    ByVal y As Double(), _
    ByVal z As Double() _
)

[C#]
public ContourDataset(
    string sname,
    double[] x,
    double[] y,
    double[] z
);

```

This constructor creates a new **ContourDataset** object defined using the supplied **SurfaceFunction** class, evaluated for the range x1,y1 to x2,y2 at intervals equal to (x2-x1)/columns for the x direction, and (y2-y1)/rows in the y direction. This forms a regular grid surface.

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal rows As Integer, _
    ByVal columns As Integer, _
    ByVal x1 As Double, _
    ByVal y1 As Double, _

```

```

    ByVal x2 As Double, _
    ByVal y2 As Double, _
    ByVal sf As SurfaceFunction _
)

[C#]
public ContourDataset(
    string sname,
    int rows,
    int columns,
    double x1,
    double y1,
    double x2,
    double y2,
    SurfaceFunction sf
);

```

The next constructor creates a dataset using the x-, y- and z-values stored in a file that uses the CSV (Comma Separated Value) format. There are two ways to organize the numeric values in the data file. If you use the COLUMN_MAJOR format, the first column represents the x-values and the second and third columns the y- and z-values. If you use the ROW_MAJOR format, the first row represents the x-values and the second and third row the y- and z-values. Use the **CSV.SetOrientation** method to initialize the csv argument for the proper data orientation.

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal csv As CSV, _
    ByVal filename As String, _
    ByVal rowskip As Integer, _
    ByVal columnskip As Integer _
)

[C#]
public ContourDataset(
    CSV csv,
    string filename,
    int rowskip,
    int columnskip
);

```

sname Specifies the name of the dataset.

grid An array, size [npoints] (or size [rows * columns]) of **Point3D** points, that specifies the xyz values of a dataset. Some of the constructors require the data points form a regular grid in the xy plane. A regular grid is one where the x-increment between adjacent x-values is fixed, as is the y-increment. The x-increment and the y-increment do not have to be the same.

rows Specifies the number of rows (in the y direction) in the regular grid. Also specifies the number of rows (or y-values) to evaluate the function over in the constructor that uses a **SurfaceFunction** argument.

<i>columns</i>	Specifies the number of columns (in the x direction) in the regular grid. Also specifies the number of columns (or y-values) to evaluate the function over in the constructor that uses a SurfaceFunction argument.
<i>npoints</i>	Specifies the number of xyz data point triplets in the grid array.
<i>x</i>	An array, size [npoints] of double that specifies the x-values of the dataset. The length of the y and z arrays must equal x.Length.
<i>y</i>	An array, size [npoints] of double that specifies the y-values of the dataset.
<i>z</i>	An array, size [npoints] of double that specifies the z-values of the dataset.
<i>x1, y1, x2, y2</i>	The SurfaceFunction sf is evaluated for the range x1,y1 to x2, y2.
<i>sf</i>	The dataset data points are created by evaluating the SurfaceFunction across the range x1,y1 to x2, y2.
<i>csv</i>	An instance of a CSV object.
<i>filename</i>	The name of the file.
<i>rowskip</i>	Skip this many rows before starting the read operation.
<i>columnskip</i>	For each row of data, skip this many columns before reading the first value from the row.

Example of creating a contour dataset from an array of Point3D

[C#]

```
int nrows=11, ncols=11;
int i, j, count=0;
double x, y, z;
double startx = -6.0, starty = -6.0;
double stepx = 12.0/(nrows-1), stepy = 12.0/(ncols-1);
Point3D []pointarray;

pointarray = new Point3D[nrows * ncols];
x = startx;
y = starty;
for (i = 0; i < nrows; i++)
```



```

{
    x = startx;
    for (j=0; j < ncols; j++)
    {
        pointarray[count] = new Point3D();
        z = 2000 + ( 950 * Math.Sin(Math.Sqrt(x*x+ y*y)));
        pointarray[count].SetLocation(x, y, z);
        x += stepx;
        count++;
    }
    y += stepy;
}

// This method triangulates data into a surface
ContourDataset dataset1 = new ContourDataset("Contour Dataset",pointarray);
// This method uses the characteristic that the data is an even spaced grid.
ContourDataset dataset2=
    new ContourDataset("Contour Dataset",pointarray, nrows, ncols);

```

[Visual Basic]

```

Dim nrows As Integer = 11
Dim ncols As Integer = 11
Dim count As Integer = 0
Dim i, j As Integer
Dim x, y, z As Double
Dim tempx, tempy As Double
Dim startx As Double = -6.0
Dim starty As Double = -6.0
Dim stepx As Double = 12.0 / (nrows - 1)
Dim stepy As Double = 12.0 / (ncols - 1)
Dim pointarray(nrows * ncols - 1) As Point3D

x = startx
y = starty
For i = 0 To nrows - 1
    x = startx
    For j = 0 To ncols - 1
        pointarray(count) = New Point3D()
        tempx = x + 1.75 * (ChartSupport.GetRandomDouble() - 0.5)
        tempy = y + 1.75 * (ChartSupport.GetRandomDouble() - 0.5)
        z = 2000 + 950 * Math.Sin(Math.Sqrt((tempx * tempx + tempy * tempy)))
        pointarray(count).SetLocation(tempx, tempy, z)
    
```

```

        x += stepx
        count += 1
    Next j
    y += stepy
Next i
dataset1 = New ContourDataset("Contour Dataset", pointarray)
` This method uses the characteristic that the data is an even spaced grid.
Dim dataset2 As ContourDataset = _
    New ContourDataset("Contour Dataset",pointarray, nrows, ncols);

```

Example of creating a contour dataset from a function

[C#]

```

ContourDataset dataset1 = null;
class ZValueFunctionClass: SurfaceFunction
{
    public override double CalcZValue(double x, double y)
    {
        double z;
        x = x + 1.75 * (ChartSupport.GetRandomDouble() - 0.5);
        y = y + 1.75 * (ChartSupport.GetRandomDouble() - 0.5);
        z = 1500 + (1500.0 * Math.Sin(Math.Sqrt(x*x+ y*y)));
        return z;
    }
}

void CreateRegularGridPolysurface()
{
    ZValueFunctionClass zValueFunction = new ZValueFunctionClass();
    dataset1 = new ContourDataset("Contour ChartDataset",11, 11,
        -6.0, -6.0, 6.0, 6.0, zValueFunction);
}

```

[Visual Basic]

```

Class ZValueFunctionClass Inherits SurfaceFunction
    Public Overrides Function CalcZValue(ByVal x As Double, ByVal y As Double) _
        As Double

        Dim z As Double
        x = x + 0.5 * (ChartSupport.GetRandomDouble() - 0.5)

```

```

    y = y + 0.5 * (ChartSupport.GetRandomDouble() - 0.5)
    z = 2000 + 950 * Math.Sin(Math.Sqrt((x * x + y * y)))
    Return z
End Function 'CalcZValue
End Class 'ZValueFunctionClass

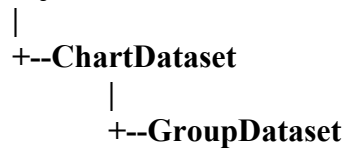
Sub CreateRegularGridPolysurface()
    Dim zValueFunction As New ZValueFunctionClass()
    dataset1 = New ContourDataset("Contour Dataset", 32, 32, _
        -7.1, -7.1, 7.1, 7.1, zValueFunction)
End Sub 'CreateRegularGridPolysurface

```

Numeric Group Dataset

Class GroupDataset

ChartObj



The **GroupDataset** class represents group data, where every x-value can have one or more y-values. The number of x-values in a group plot is referred to as the number of columns or as **numberDatapoints** and the number of y-values *for each x-value* is referred to as the number of rows, or **numberGroups**. Think of spreadsheet file that looks like

x-values	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
y-values group #0	y[0,0]	y[0,1]	y[0,2]	y[0,3]	y[0,4]	y[0,5]
y-values group #1	y[1,0]	y[1,1]	y[1,2]	y[1,3]	y[1,4]	y[1,5]
y-values group #2	y[2,0]	y[2,1]	y[2,2]	y[2,3]	y[2,4]	y[2,5]

number of x-values = **numberDatapoints** = numberColumns = 6

number of y-values for each x-value = **numberGroups** = numberRows = 3

This would be the ROW_MAJOR format if the data were stored in a CSV file.

GroupDataset constructors

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As Double(), _
    ByVal y As Double(), _
)

Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As Double(), _
    ByVal y As Double(), _
    ByVal zvalue As Double, _
    ByVal depthvalue As Double _
)

[C#]
public GroupDataset(
    string sname,
    double[] x,
    double[,] y
);

public GroupDataset(
    string sname,
    double[] x,
    double[,] y,
    double zvalue,
    double depthvalue
);

```

<i>sname</i>	Specifies the name of the dataset.
<i>x</i>	An array that specifies the x-values of a group dataset. The length of the x array sets the number of columns for the group dataset.
<i>y</i>	An array that specifies the y-values of a group dataset where y has the dimensions [number of rows, number of columns]. The number of rows in the y-array sets the number of groups in the group dataset. The number of columns in the y-array must match the length of the x-array.
<i>zvalue</i>	Sets the ImplicitZValue for the dataset. The default value is 0.0;
<i>depthvalue</i>	Sets the ImplicitDepthValue for the dataset. The default value is 0.1.

The number of columns in the group dataset is the value of `x.Length` property. The number of columns in the y array must match the length of the x array and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

The next constructor creates a dataset using the x- and y-values stored in a file that uses the CSV (Comma Separated Value) format. There are two ways to organize the numeric values in the data file. If you use the COLUMN_MAJOR format, the first column represents the x-values and subsequent columns represent the y-values, where each column is a group. If you use the ROW_MAJOR format, the first row represents the x-values and subsequent rows represent the y-values, where each row is a group. Use the **CSV.SetOrientation** method to initialize the csv argument for the proper data orientation.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal csv As CSV, _
    ByVal filename As String, _
    ByVal rowskip As Integer, _
    ByVal columnskip As Integer _
)

[C#]
public GroupDataset(
    CSV csv,
    string filename,
    int rowskip,
    int columnskip
);
```

<i>csv</i>	An instance of a CSV object.
<i>filename</i>	The name of the file.
<i>rowskip</i>	Skip this many rows before starting the read operation.
<i>columnskip</i>	For each row of data, skip this many columns before reading the first value from the row.

You can retrieve references to the internal arrays used to store the data using the **GroupDataset** methods **GetXData** and **GetGroupData**. Change the values in the data arrays using these references. You can also modify a point at a time using one of the **SetYDataValue** and **SetXDataValue** methods. If you need to add new points to dataset, increasing its size, use one of the **AddGroupDataPoints**, or **InsertGroupDataPoints** methods. Delete data points using the **DeleteDataPoint** method. In order to see the modified dataset, force the graph to redraw using **ChartView.UpdateDraw** method.

Example of creating a group datasets from numeric arrays

```
[C#]

double []x1= {10,20,30,40,50};
double [,]y1 = {{ 9,-21, 20,40,30},
```

90 Chart Datasets

```
        { 55,15,35,10,56},  
        {15,25,15,30,40}};  
  
double depth = 0.5;  
double zvalue = 0.5;  
GroupDataset Dataset11 = new GroupDataset("First",x1, y1, zvalue, depth);
```

[Visual Basic]

```
Dim x1() As Double = {10, 20, 30, 40, 50}  
Dim y1(,) As Double = {{9, -21, 20, 40, 30}, _  
                        {55, 15, 35, 10, 56}, _  
                        {15, 25, 15, 30, 40}}  
  
Dim depth As Double = 0.5  
Dim zvalue As Double = 0.5  
Dim Dataset1 As GroupDataset = New GroupDataset("First", x1, y1, xvalue, depth)
```

Example of creating a group datasets from a CSV file

[C#]

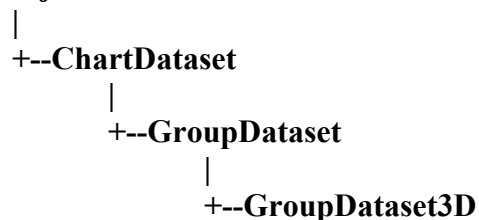
```
CSV csvDataFile = new CSV();  
GroupDataset Dataset1 =  
    new GroupDataset(csvDataFile,"GroupDataset.csv",0,0);  
// Write out dataset as a CSV file under a different file name  
Dataset1.WriteGroupDataset (csvDataFile,"GroupDataset2.csv");
```

[Visual Basic]

```
Dim csvDataFile As CSV = New CSV()  
Dim Dataset1 As GroupDataset = _  
    New GroupDataset(csvDataFile, "GroupDataset.csv", 0, 0)  
' Write out dataset as a CSV file under a different file name  
Dataset1.WriteGroupDataset(csvDataFile, "GroupDataset2.csv")
```

Class GroupDataset3D

ChartObj



The **GroupDataset3D** class represents group data, where every x-value has one or more y-values, and a z-value. The number of x-values in a group plot is referred to as the number of columns or as **numberDatapoints** and the number of y-values *for each x-value* is referred to as the number of rows, or **numberGroups**. Think of spreadsheet file that looks like

x-values	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
y-values group #0	y[0,0]	y[0,1]	y[0,2]	y[0,3]	y[0,4]	y[0,5]
y-values group #1	y[1,0]	y[1,1]	y[1,2]	y[1,3]	y[1,4]	y[1,5]
y-values group #2	y[2,0]	y[2,1]	y[2,2]	y[2,3]	y[2,4]	y[2,5]
z-values	z[0]	z[1]	z[2]	z[3]	z[4]	z[5]

number of x-values = **numberDatapoints** = numberColumns = 6

number of y-values for each x-value = **numberGroups** = numberOfRows = 3

This would be the ROW_MAJOR format if the data were stored in a CSV file.

GroupDataset constructors

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As Double(), _
    ByVal y As Double(), _
    ByVal z As Double() _
)

Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As Double(), _
    ByVal y As Double(), _
    ByVal z As Double(), _
    ByVal depthvalue As Double _
)

[C#]
public GroupDataset(
    string sname,
    double[] x,
    double[,] y
    double[] z
);
```

```
public GroupDataset(
    string sname,
    double[] x,
    double[,] y,
    double[] z
    double depthvalue
);
```

<i>sname</i>	Specifies the name of the dataset.
<i>x</i>	An array that specifies the x-values of a group dataset. The length of the x array sets the number of columns for the group dataset.
<i>y</i>	An array that specifies the y-values of a group dataset where y has the dimensions [number of rows, number of columns]. The number of rows in the y-array sets the number of groups in the group dataset. The number of columns in the y-array must match the length of the x-array.
<i>z</i>	An array that specifies the z-values of a group dataset. The length of the z array sets the number of columns for the group dataset
<i>depthvalue</i>	Sets the ImplicitDepthValue for the dataset. The default value is 0.1.

The number of columns in the group dataset is the value of `x.Length` property. The number of columns in the y array must match the length of the x array and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

The next constructor creates a dataset using the x- and y-values stored in a file that uses the CSV (Comma Separated Value) format. There are two ways to organize the numeric values in the data file. If you use the COLUMN_MAJOR format, the first column represents the x-values and subsequent columns represent the y-values, where each column is a group. If you use the ROW_MAJOR format, the first row represents the x-values and subsequent rows represent the y-values, where each row is a group. Use the **CSV.SetOrientation** method to initialize the csv argument for the proper data orientation.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal csv As CSV, _
    ByVal filename As String, _
    ByVal rowskip As Integer, _
    ByVal columnskip As Integer _
)

[C#]
public GroupDataset3D(
    CSV csv,
```



```

    string filename,
    int rowskip,
    int columnskip
);

```

<i>csv</i>	An instance of a CSV object.
<i>filename</i>	The name of the file.
<i>rowskip</i>	Skip this many rows before starting the read operation.
<i>columnskip</i>	For each row of data, skip this many columns before reading the first value from the row.

You can retrieve references to the internal arrays used to store the data using the **GroupDataset3D** methods **GetXData** and **GetGroupData**. Change the values in the data arrays using these references. You can also modify a point at a time using one of the **SetYDataValue** and **SetXDataValue** methods. If you need to add new points to dataset, increasing its size, use one of the **AddGroupDataPoints**, or **InsertGroupDataPoints** methods. Delete data points using the **DeleteDataPoint** method. In order to see the modified dataset, force the graph to redraw using **ChartView.UpdateDraw** method.

Example of creating a group datasets from numeric arrays

[C#]

```

double []x1= {10,20,30,40,50};
double [,]y1 = {{ 9,-21, 20,40,30},
                { 55,15,35,10,56},
                {15,25,15,30,40}};
double []z1= {0.1,0.2,0.3,0.4,0.5};
double depth = 0.5;
GroupDataset3D Dataset11 = new GroupDataset3D("First",x1, y1, z1, depth);

```

[Visual Basic]

```

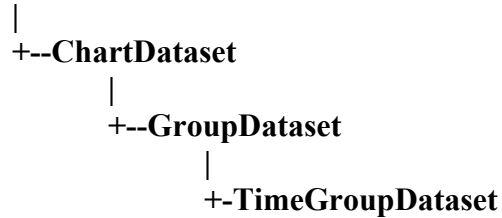
Dim x1 As Double() = {10, 20, 30, 40, 50}
Dim y1 As Double(,) = {{9, -21, 20, 40, 30}, _
                       {55, 15, 35, 10, 56}, _
                       {15, 25, 15, 30, 40}}
Dim z1 As Double() = {0.1, 0.2, 0.3, 0.4, 0.5}
Dim depth As Double = 0.5
Dim Dataset1 As New GroupDataset3D("First", x1, y1, z1, depth)

```

Date/Time Group Dataset

Class TimeGroupDataset

ChartObj



The **TimeGroupDataset** class represents group data, where every x-value can have one or more y-values. It uses **ChartCalendar** dates as the x-values, and floating point numbers as the y-values. **ChartCalendar** values are actually stored internally as their equivalent millisecond values. The **TimeGroupDataset** class adds a large number of methods to the **GroupDataset** class that make it easy to create and modify datasets that use **ChartCalendar** values.

This constructor creates a new group **TimeGroupDataset** object where the x-values are **ChartCalendar** values and the y-values are floating point numbers.

TimeGroupDataset constructors

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As ChartCalendar(), _
    ByVal y As Double(), _
)

Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As ChartCalendar(), _
    ByVal y As Double(), _
    ByVal zvalue As Double, _
    ByVal depthvalue As Double _
)

[C#]
public TimeGroupDataset(
    string sname,
    ChartCalendar[] x,
    double[,] y
);

public TimeGroupDataset(
    string sname,
    ChartCalendar[] x,
    double[,] y,
    double zvalue,
    double depthvalue
);
  
```

sname Specifies the name of the dataset.

<i>x</i>	An array of ChartCalendar dates, that specifies the x-values of a dataset. The length of the x array sets the number of columns for the group dataset.
<i>y</i>	An array that specifies the y-values of a group dataset where y has the dimensions [number of rows, number of columns]. The number of rows in the y-array sets the number of groups in the group dataset. The number of columns in the y-array must match the length of the x-array.
<i>zvalue</i>	Sets the ImplicitZValue for the dataset. The default value is 0.0;
<i>depthvalue</i>	Sets the ImplicitDepthValue for the dataset. The default value is 0.1.

The number of columns in the group dataset is the value of `x.Length` property. The number of columns in the y array must match the length of the x array and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal csv As CSV, _
    ByVal filename As String, _
    ByVal rowskip As Integer, _
    ByVal columnskip As Integer _
)

[C#]
public TimeGroupDataset(
    CSV csv,
    string filename,
    int rowskip,
    int columnskip
);
```

<i>csv</i>	An instance of a CSV object.
<i>filename</i>	The name of the file.
<i>rowskip</i>	Skip this many rows before starting the read operation.
<i>columnskip</i>	For each row of data, skip this many columns before reading the

There are two ways to organize the numeric values in the data file. If you use the COLUMN_MAJOR format, the first column represents the time values and subsequent

columns represent the y-values, where each column is a group. If you use the ROW_MAJOR format, the first row represents the time values and subsequent rows represent the y-values, where each row is a group. Use the **CSV.SetOrientation** method to initialize the csv argument for the proper data orientation.

A **DateTimeFormatInfo** object, and a date time format string, in the **CSV** class, control the interpretation of the **ChartCalendar** values. The format in the file must match the format specified for the **CSV** class. The underlying conversion mechanism calls the **DateTime.ToString(String formatstring, DateTimeFormatInfo info)** method for the conversion. The default format for the date time *formatstring* object is "M/dd/yy". Call the **SetDateTimeFormatString** method to change the default date time format. See the documentation for the .Net **DateTime.ToString** method to figure out the various formatting options for the date time format string. If you are into internationalization (and difficult to understand .Net documentation), you can also create your own **DateTimeFormatInfo** object, installing it in the **CSV** object using **CSV.SetTimeDateFormat** method. The date time format string and the **DateTimeFormatInfo** object apply to both CSV files used for input, and CSV files used for output. If an attempt is made to read date/time values that do not match the desired format, the data values are set to invalid date/time values.

You can retrieve a *copy* of the date time data using the **TimeGroupDataset.GetTimeXData** method. It returns an array of **ChartCalendar** objects, and it is *not* a reference to the underlying data. The underlying data is stored as double values that represent the millisecond equivalent of the date time values. The **TimeGroupDataset** **GetXData** and **GetYData** methods return references to the underlying data. You can also modify a point at a time using one of the **TimeGroupDataset**, **SetTimeXDataValue** and **SetYDataValue** methods. If you need to add new points to dataset, increasing its size, use one of the **AddTimeGroupDataPoints**, or **InsertTimeGroupDataPoints** methods. Delete data points using the **DeleteDataPoint** method. In order to see the modified dataset, force the graph to redraw using **ChartView.UpdateDraw** method.

Example of creating a group time datasets

[C#]

```
int nNumPnts = 50, nNumGroups = 4;
int weekmode = ChartObj.WEEK_5D;
ChartCalendar []xValues= new ChartCalendar[nNumPnts];
double [,]stockPriceData = new double[nNumGroups,nNumPnts];
double minval=0.0, maxval = 0.0;
int i;

ChartCalendar currentdate = new ChartCalendar();
```

```

ChartCalendar.SetTOD(currentdate,0,0,1);
currentdate = ChartCalendar.CalendarDaysAdd(currentdate, 1, weekmode);
    // Make sure not to start on a weekend
xValues[0] = (ChartCalendar) currentdate.Clone();
currentdate = ChartCalendar.CalendarDaysAdd(currentdate, 1, weekmode);
stockPriceData[3,0] = 25; // close
stockPriceData[0,0] = 25; // open
stockPriceData[1,0] = 26; // high
stockPriceData[2,0] = 24; // low

for (i=1; i < nNumPnts; i++)
{
    xValues[i] = (ChartCalendar) currentdate.Clone();
    stockPriceData[3,i] += stockPriceData[3,i-1] +
        3 * (0.52 - ChartSupport.GetRandomDouble()); // close
    stockPriceData[0,i] += stockPriceData[3,i] +
        2 * (0.5 - ChartSupport.GetRandomDouble()); // open
    minval = Math.Min(stockPriceData[3,i], stockPriceData[0,i]);
    maxval = Math.Max(stockPriceData[3,i], stockPriceData[0,i]);
    stockPriceData[1,i] = maxval + 1.5 * ChartSupport.GetRandomDouble(); // high
    stockPriceData[2,i] = minval - 1.5 * ChartSupport.GetRandomDouble(); // low
    currentdate = ChartCalendar.CalendarDaysAdd(currentdate, 1, weekmode);
}
double zvalue = 0.0;
double depthvalue = 0.5;
TimeGroupDataset Dataset1 = new
    TimeGroupDataset("Stock Data",xValues,stockPriceData, zvalue, depthvalue);

```

[Visual Basic]

```

Dim nNumPnts As Integer = 50
Dim nNumGroups As Integer = 4
Dim weekmode As Integer = ChartObj.WEEK_5D
Dim xValues(nNumPnts - 1) As ChartCalendar
Dim stockPriceData(nNumGroups - 1, nNumPnts - 1) As Double
Dim minval As Double = 0.0
Dim maxval As Double = 0.0
Dim i As Integer
Dim currentdate As New ChartCalendar()
' Make sure not to start on a weekend

```

98 Chart Datasets

```
currentdate = ChartCalendar.CalendarDaysAdd(currentdate, 1, weekmode)
xValues(0) = currentdate.Clone()
currentdate = ChartCalendar.CalendarDaysAdd(currentdate, 1, weekmode)
stockPriceData(3, 0) = 25 ' close
stockPriceData(0, 0) = 25 ' open
stockPriceData(1, 0) = 26 ' high
stockPriceData(2, 0) = 24 ' low
For i = 1 To nNumPnts - 1
    xValues(i) = currentdate.Clone()
    stockPriceData(3, i) += stockPriceData(3, i - 1) + _
        3 * (0.52 - ChartSupport.GetRandomDouble()) ' close
    stockPriceData(0, i) += stockPriceData(3, i) + _
        2 * (0.5 - ChartSupport.GetRandomDouble()) ' open
    minval = Math.Min(stockPriceData(3, i), stockPriceData(0, i))
    maxval = Math.Max(stockPriceData(3, i), stockPriceData(0, i))
    stockPriceData(1, i) = maxval + 1.5 * ChartSupport.GetRandomDouble() ' high
    stockPriceData(2, i) = minval - 1.5 * ChartSupport.GetRandomDouble() ' low
    currentdate = ChartCalendar.CalendarDaysAdd(currentdate, 1, weekmode)
Next i
Dim zvalue As Double = 0.0
Dim depthvalue As Double = 0.5
Dim Dataset1 As New
    TimeGroupDataset("Stock Data", xValues, stockPriceData, zvalue, depthvalue)
```

Example of creating a simple time datasets from a CSV file

[C#]

```
CSV csvDataFile = new CSV();

TimeGroupDataset Dataset1 =
    new TimeGroupDataset(csvDataFile, "TimeGroupDataset.csv", 0, 0);
Dataset1.ImplicitZValue = 0.5;
Dataset1.ImplicitDepthValue = 0.5;
// Write out dataset as a CSV file under a different file name
Dataset1.WriteTimeGroupDataset (csvDataFile, "TimeGroupDataset2.csv");
```

[Visual Basic]

```
'Default time date format is "M/dd/yyyy"
Dim csvDataFile As CSV = New CSV()
```

```

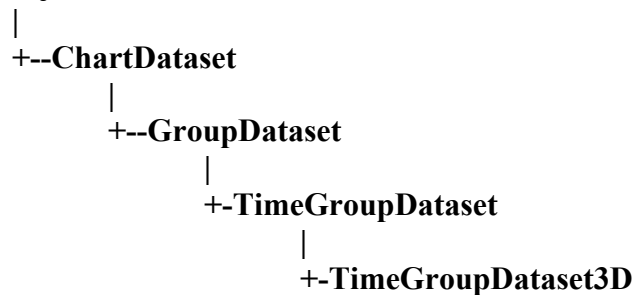
'Create a dataset based on a previously saved csv file
Dim Dataset1 As TimeGroupDataset = _
    New TimeGroupDataset (csvDataFile, "TimeGroupDataset.csv", 0, 0)
Dataset1.ImplicitZValue = 0.5
Dataset1.ImplicitDepthValue = 0.5

'Write out dataset as a CVS file
Dataset1.WriteTimeGroupDataset (csvDataFile, " TimeGroupDataset1.csv")
' Read it back in just as a test
Dataset1.ReadTimeGroupDataset(csvDataFile, " TimeGroupDataset1.csv", 0, 0)

```

Class TimeGroupDataset3D

ChartObj



The **TimeGroupDataset3D** represents group data, where every x-value has one or more y-values, and a z-value. It uses **ChartCalendar** dates as the x-values, and floating point numbers as the y- and z-values. **ChartCalendar** values are actually stored internally as their equivalent millisecond values. The **TimeGroupDataset3D** class adds a large number of methods to the **GroupDataset** class that make it easy to create and modify datasets that use **ChartCalendar** values.

This constructor creates a new group **TimeGroupDataset3D** object where the x-values are **ChartCalendar** values and the y-values are floating point numbers.

TimeGroupDataset3D constructors

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As ChartCalendar(), _
    ByVal y As Double(), _
    ByVal z As Double(), _
)

Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As ChartCalendar(), _
    ByVal y As Double(), _

```

100 Chart Datasets

```
ByVal z As Double (), _  
ByVal depthvalue As Double _  
)  
  
[C#]  
public TimeGroupDataset3D(  
    string sname,  
    ChartCalendar[] x,  
    double[,] y,  
    double[] z  
)  
;  
  
public TimeGroupDataset3D(  
    string sname,  
    ChartCalendar[] x,  
    double[,] y,  
    double[] z,  
    double depthvalue  
)  
;
```

<i>sname</i>	Specifies the name of the dataset.
<i>x</i>	An array of ChartCalendar dates, that specifies the x-values of a dataset. The length of the x array sets the number of columns for the group dataset.
<i>y</i>	An array that specifies the y-values of a group dataset where y has the dimensions [number of rows, number of columns]. The number of rows in the y-array sets the number of groups in the group dataset. The number of columns in the y-array must match the length of the x-array.
<i>z</i>	An that specifies the x-values of a dataset. The length of the x array sets the number of columns for the group dataset.
<i>depthvalue</i>	Sets the ImplicitDepthValue for the dataset. The default value is 0.1.

The number of columns in the group dataset is the value of x.Length property. The number of columns in the y array must match the length of the x array and every element must be initialized to a valid value. All values in the arrays are plotted. If the data is outside of the current chart scale the values will be clipped.

```
[Visual Basic]  
Overloads Public Sub New( _  
    ByVal csv As CSV, _  
    ByVal filename As String, _  
    ByVal rowskip As Integer, _  
    ByVal columnskip As Integer _  
)
```



```
[C#]
public TimeGroupDataset3D(
    CSV csv,
    string filename,
    int rowskip,
    int columnskip
);
```

<i>csv</i>	An instance of a CSV object.
<i>filename</i>	The name of the file.
<i>rowskip</i>	Skip this many rows before starting the read operation.
<i>columnskip</i>	For each row of data, skip this many columns before reading the

There are two ways to organize the numeric values in the data file. If you use the **COLUMN_MAJOR** format, the first column represents the time values and subsequent columns represent the y-values, where each column is a group. If you use the **ROW_MAJOR** format, the first row represents the time values and subsequent rows represent the y-values, where each row is a group. Use the **CSV.SetOrientation** method to initialize the *csv* argument for the proper data orientation.

A **DateTimeFormatInfo** object, and a date time format string, in the **CSV** class, control the interpretation of the **ChartCalendar** values. The format in the file must match the format specified for the **CSV** class. The underlying conversion mechanism calls the **DateTime.ToString(String formatstring, DateTimeFormatInfo info)** method for the conversion. The default format for the date time *formatstring* object is "M/dd/yy". Call the **SetDateTimeFormatString** method to change the default date time format. See the documentation for the .Net **DateTime.ToString** method to figure out the various formatting options for the date time format string. If you are into internationalization (and difficult to understand .Net documentation), you can also create your own **DateTimeFormatInfo** object, installing it in the **CSV** object using **CSV.SetTimeDateFormat** method. The date time format string and the **DateTimeFormatInfo** object apply to both CSV files used for input, and CSV files used for output. If an attempt is made to read date/time values that do not match the desired format, the data values are set to invalid date/time values.

You can retrieve a *copy* of the date time data using the **TimeGroupDataset3D.GetTimeXData** method. It returns an array of **ChartCalendar** objects, and it is *not* a reference to the underlying data. The underlying data is stored as double values that represent the millisecond equivalent of the date time values. The **TimeGroupDataset3D.GetXData** and **GetYData** methods return references to the underlying data. You can also modify a point at a time using one of the **TimeGroupDataset3D.SetTimeXDataValue** and **SetYDataValue** methods. If you need to add new points to dataset, increasing its size, use one of the

AddTimeGroupDataPoints, or **InsertTimeGroupDataPoints** methods. Delete data points using the **DeleteDataPoint** method. In order to see the modified dataset, force the graph to redraw using **ChartView.UpdateDraw** method.

Example of creating a group time datasets

[C#]

```
int nNumPnts = 50, nNumGroups = 4;
int weekmode = ChartObj.WEEK_5D;
ChartCalendar []xValues= new ChartCalendar[nNumPnts];
double [,]stockPriceData = new double[nNumGroups,nNumPnts];
double []zValues= new double[nNumPnts];
double minval=0.0, maxval = 0.0;
int i;

ChartCalendar currentdate = new ChartCalendar();
ChartCalendar.SetTOD(currentdate,0,0,1);
currentdate = ChartCalendar.CalendarDaysAdd(currentdate, 1, weekmode);
    // Make sure not to start on a weekend
xValues[0] = (ChartCalendar) currentdate.Clone();
zValues[0] = 0.3;

currentdate = ChartCalendar.CalendarDaysAdd(currentdate, 1, weekmode);
stockPriceData[3,0] = 25; // close
stockPriceData[0,0] = 25; // open
stockPriceData[1,0] = 26; // high
stockPriceData[2,0] = 24; // low

for (i=1; i < nNumPnts; i++)
{
    xValues[i] = (ChartCalendar) currentdate.Clone();
    stockPriceData[3,i] += stockPriceData[3,i-1] +
        3 * (0.52 - ChartSupport.GetRandomDouble()); // close
    stockPriceData[0,i] += stockPriceData[3,i] +
        2 * (0.5 - ChartSupport.GetRandomDouble()); // open
    minval = Math.Min(stockPriceData[3,i], stockPriceData[0,i]);
    maxval = Math.Max(stockPriceData[3,i], stockPriceData[0,i]);
    stockPriceData[1,i] = maxval + 1.5 * ChartSupport.GetRandomDouble(); // high
    stockPriceData[2,i] = minval - 1.5 * ChartSupport.GetRandomDouble(); // low
    currentdate = ChartCalendar.CalendarDaysAdd(currentdate, 1, weekmode);
}
```

```

        zValues[i] = 0.3;
    }
    double depthvalue = 0.5;
    TimeGroupDataset3D Dataset1 = new
        TimeGroupDataset3D("Stock Data",xValues,stockPriceData, zValues,
        depthvalue);

```

[Visual Basic]

```

Dim nNumPnts As Integer = 50
Dim nNumGroups As Integer = 4
Dim weekmode As Integer = ChartObj.WEEK_5D
Dim xValues(nNumPnts - 1) As ChartCalendar
Dim zValues(nNumPnts - 1) As Double
Dim stockPriceData(nNumGroups - 1, nNumPnts - 1) As Double
Dim minval As Double = 0.0
Dim maxval As Double = 0.0
Dim i As Integer
Dim currentdate As New ChartCalendar()
' Make sure not to start on a weekend
currentdate = ChartCalendar.CalendarDaysAdd(currentdate, 1, weekmode)
xValues(0) = currentdate.Clone()
zValues(0) = 0.3
currentdate = ChartCalendar.CalendarDaysAdd(currentdate, 1, weekmode)
stockPriceData(3, 0) = 25 ' close
stockPriceData(0, 0) = 25 ' open
stockPriceData(1, 0) = 26 ' high
stockPriceData(2, 0) = 24 ' low
For i = 1 To nNumPnts - 1
    xValues(i) = currentdate.Clone()
    stockPriceData(3, i) += stockPriceData(3, i - 1) + _
        3 * (0.52 - ChartSupport.GetRandomDouble()) ' close
    stockPriceData(0, i) += stockPriceData(3, i) + _
        2 * (0.5 - ChartSupport.GetRandomDouble()) ' open
    minval = Math.Min(stockPriceData(3, i), stockPriceData(0, i))
    maxval = Math.Max(stockPriceData(3, i), stockPriceData(0, i))
    stockPriceData(1, i) = maxval + 1.5 * ChartSupport.GetRandomDouble() ' high
    stockPriceData(2, i) = minval - 1.5 * ChartSupport.GetRandomDouble() ' low
    currentdate = ChartCalendar.CalendarDaysAdd(currentdate, 1, weekmode)
    zValues(i) = 0.3;
Next i

```

104 *Chart Datasets*

```
Dim depthvalue As Double = 0.5
Dim Dataset1 As New _
    TimeGroupDataset3D("Stock Data", xValues, stockPriceData, zValues, depthvalue)
```

4. Scaling and Coordinate Systems

ChartScale

LinearScale

LogScale

TimeScale

Transform3D

UserCoordinates

WorldCoordinates

WorkingCoordinates

PhysicalCoordinates

CartesianCoordinates

TimeCoordinates

The starting point for all drawing in a window is the .Net 2D device coordinate system. The coordinate system uses a default device resolution of the underlying .Net window. A .Net window maintains a viewport for the client area of the window, controlling the position and size of the drawing area in the window. Graphics output is clipped to the viewport, preventing graphics output in one window from over-writing graphics in another window. The user coordinate system for the window starts at (0,0) in the upper left corner and extends in the positive direction down and to the right.

This software maps a 3D physical coordinate system on the .Net 2D device coordinate system. The 3D transformation pipeline looks something like this:

3D Physical Coordinates -> 3D User/Device Coordinates -> 2D Screen Coordinates

The position and size of all 3D chart objects are defined using one of the 3D physical coordinate systems (**CartesianCoordinates** or **TimeCoordinates**). A chart object is broken into component lines and polygons, the 3D coordinates of which are transformed through scaling operations into 3D user coordinates. User coordinates are based on the device coordinate dimensions of the window the chart is displayed in. The 3D user coordinates are then processed using standard 3D transformation techniques (4x4 homogeneous matrix conversions), rotating and translating the coordinates into a flat 2D projection of the 3D image. The resulting screen coordinates match the .Net device coordinates of the underlying .Net window.

Plot area volume and the graph area volume

The *plot area volume* of a graph is the 3D volume where the plot data objects (line plots, bar plots, etc.) are drawn. The *graph area volume* is the entire volume of the chart window. The graph area volume includes the plot area volume as a subset. Usually, the plot area volume is smaller than the graph area volume and resides roughly centered in

the graph area volume. The border around the plot area volume is sized large enough to display the axis tick mark labels, axis titles, legends, chart titles, footers, and any other object in the graph. Create a physical coordinate system for a chart, and you are setting the minimum and maximum values for the x, y, and z dimensions of the plot area volume.

Most chart objects require access to the chart coordinate system for proper positioning in the chart window. Some chart objects, axis objects in particular, often reside on the edge or outside of the plot area volume. Important parts of the axis, the tick marks and tick mark labels, are usually outside of the plot area volume. The tick marks and tick mark labels must align perfectly with the coordinate system inside the plot area volume.

There are many different techniques to align the coordinate system inside the plot area volume with the coordinate system used in drawing chart objects outside of the plot area volume. One technique is to maintain the physical coordinate system inside the plot area volume, and use a normalized coordinate system for the graph area volume. Whenever a chart object in the graph area volume, a y-axis tick mark for example, needs to be aligned with the coordinate system inside the plot area volume, the software converts the tick mark placement value from physical coordinates to normalized coordinates using standardized coordinate conversion routines. The drawback of this technique is what I will call the “odd pixel problem”. The odd pixel problem shows up when you try map physical, normalized and user coordinate systems based on floating point numbers onto a pixel coordinate system using an integer coordinate system. Unless the corners of the plot area volume fall on exact pixel boundaries, converting from plot area volume coordinates to graph area volume coordinates, once translated to pixels, can be up to one pixel off.

The alternative technique, used in this software library, is to use a single coordinate system. The physical coordinate system defined for the plot area volume is extended in all size directions – left, right, top, bottom, out of screen, into screen. It is extended so that the physical coordinates of the six corners of the plot area volume remain unchanged. The six corners of the graph area volume are assigned calculated, physical coordinate values so that when it is overlaid on to the plot area volume there is an exact 1:1 correspondence for all points inside the plot area volume. Once this calculation is made, there is no need to use the physical coordinate system assigned to the plot area volume. Instead, the physical coordinate system of the graph area volume is used instead. Chart axes objects and plotted data always align because they are plotted using the exact same physical coordinate system. The only difference is that plotted data is clipped to the plot area volume while axes objects are not clipped.

For example, assume a graph area volume with the dimensions of 400x400x400 units, and a plot area volume with the dimensions 200x200x200 centered inside the graph area volume. This implies that there is a 100 unit boundary around all six sides of the plot area volume. The desired chart uses a physical coordinate system of (0, 0,0,100,100,100). These coordinates apply to the plot area volume. Instead of using the plot area volume coordinate system, the coordinate system (-50,-50,-50,150,150,150) is calculated and used to scale the graph area volume. This does not guarantee that any point plotted in plot area volume coordinate system will always map to the exact same pixel as the same point

plotted in the graph coordinate system, the odd pixel problem still exists. We avoid the odd pixel problem by never plotting points using the plot area volume coordinate system, using only the graph area volume coordinate system instead. The calculated physical coordinate system applied to the graph area volume is referred to as the *working* coordinate system.

Coordinate Systems

A **QCChart3D** for *.Net* library uses other coordinate systems mapped onto the default *.Net* device coordinate system. These other coordinate systems include user coordinate, world coordinates, working coordinates, and physical coordinates.

User Coordinates

The **UserCoordinates** class manages a simple 3D viewport drawing system, and manages the transformation of 3D user coordinates into the 2D device coordinate system of the *.Net* System.Drawing classes.

World Coordinates

The **WorldCoordinates** class maps a linear, double based, coordinate system onto the integer based user coordinate system of the **UserCoordinates** class. Where the underlying user coordinate system may have an integer range (for example 0-400, 0-300, 0-100 units), the world coordinate system is able to map this to a completely arbitrary, double based, linear range (for example (0.0 to 10.0, 0.0 to 10.0, 0.0 to 1.0) . The world coordinate system applies to the entire graph window, and not just the plot area volume.

Working Coordinates

The **WorkingCoordinates** class manages a working coordinate system that maps the physical coordinate system of the plot area volume into a linear, world coordinate system applied to the whole viewport. For example, if the desired chart plot area volume uses a physical coordinate system of (0.0, 0.0, 0.0, 100.0, 100.0, 1.0) and the plot area volume is centered in the graph area volume with the plot area volume $\frac{1}{2}$ the width and height of the graph area volume, then the coordinate system (-50, -50, -0.5, 150, 150, 1.5) is calculated and used to scale the graph area volume. The **WorkingCoordinates** class uses the underlying **WorldCoordinates** class to scale the viewport to the final world coordinates scale.

Physical Coordinates

The **PhysicalCoordinates** abstract class is responsible for mapping the plot area volume coordinate system (whether it is linear, logarithmic, and date/time, continuous or discontinuous) into a continuous linear coordinate system. It uses the **WorkingCoordinates** class to map this plot area volume coordinate system to the entire viewport. The **PhysicalCoordinates** system uses independent scale objects, derived from **ChartScale**, to manage coordinate conversions for the x-, y- and z-dimensions. This way

the x-coordinate can use one coordinate conversion object (**LinearScale**, **LogScale**, **TimeScale**) and the y-coordinate another.

There are two concrete implementations of the **PhysicalCoordinates** class:

CartesianCoordinates and, **TimeCoordinates**. Use the **CartesianCoordinates** class for any combination of linear and logarithmic scaling for the x- and y-coordinate. Use the **TimeCoordinates** class when you want a time scale for the x-coordinate and a linear or logarithmic scale for the y-coordinate.

Normalized coordinates

Normalized coordinates are a special case of linear physical coordinates, where the linear physical scale (0.0 - 1.0, 0.0 - 1.0, 0.0 - 1.0) is applied to either the graph area volume, or the plot area volume of the chart. *Graph normalized coordinates* maps the upper left corner of the graph window to the xyz coordinates (0.0,0.0,0.0) and the lower right corner of the graph area volume to the xyz coordinate (1.0,1.0,0.0). *Plot normalized coordinates* maps the upper left corner of the plot area volume to the xyz coordinates (0.0,0.0,0.0) and the lower right corner of the plot area volume to the xyz coordinates (1.0,1.0,0.0).

Chart Scaling and the .Net Drawing2D.Matrix

The **System.Drawing.Drawing2D.Matrix** class can handle a large number of scaling operations, including linear operations that look at first glance capable of transforming chart physical coordinates to .Net device coordinates. Why not use the **Drawing2D.Matrix** class?

There are at least four reasons. First, the **Drawing2D.Matrix** class not only scales the endpoints of lines, which is what we want, it also has the unwanted side effect of scaling the thickness of lines. It is important that the thickness of lines remain unchanged, regardless of the desired physical coordinate system. Second, the **Drawing2D.Matrix** transform does not handle logarithmic scaling, needed for physical coordinate systems that use logarithmic or semi-logarithmic scaling. Third, the **Drawing2D.Matrix** class does not support discontinuous scales as needed date/time scale and coordinate classes. Fourth and last, a point transformed using a **Drawing2D.Matrix** undergoes a time consuming, relatively complicated matrix math calculation involving a 3x3 matrix. For these reasons, the coordinate conversion routines do not use the **Drawing2D.Matrix** class. Other chart classes use the **Drawing2D.Matrix** class for scaling and rotation of graph objects such as shapes, images, and text.

A copy of a concrete instance of the **PhysicalCoordinates** class is stored in every **GraphObj** derived object. This includes all axes, plot objects, text objects and data markers. When a chart object draws itself, it uses the viewport and the physical coordinate system stored in its instance of a **PhysicalCoordinates** class. A chart can have one or more instances of the **PhysicalCoordinates** since a single chart can plot data against one or more physical coordinate systems.

Important numeric considerations

Value limiting

A chart should be scaleable to any numeric range that a user wants to plot. This incorporates the entire range of floating point numbers support under .Net. A range of $\pm 10^{-30}$ is just as valid as a range of $\pm 10^{30}$, even though there is 60 orders of magnitude of difference. A user can attempt to plot data with an extremely large dynamic range (the $\pm 10^{30}$ range) in a chart scaled for an extremely small range (the $\pm 10^{-30}$ range). The resulting user coordinate values resulting from such an extreme case can easily exceed the numeric range supported by the plotting functions.

Bad value checking

Invalid data often finds its way into chart. Invalid data can take many different forms. The most obvious is the introduction of numeric values that do not fit the .Net floating point format. These types of numbers are often found in databases and representing non-initialized or improperly initialized data. .Net cannot include an invalid floating point number in a calculation, so it is best to try and avoid them. Another type of invalid data is data that is a valid floating point number, but is never less considered invalid by the user. Often when data is outside of a predetermined range, it is invalid. Mark a data value in a dataset invalid using the **ChartDataset.SetValidData** method. If a data value equals `Double.MAX_VALUE`, it is also considered invalid.

Taking the logarithm of 0 or a negative number

If a charting package is capable of logarithmic and semi-logarithmic plotting, it must be protected against the error condition of taking the log of any number ≤ 0.0 . This often happens when a chart is initially setup with a linear scale, with a minimum physical coordinate value of 0.0 and a maximum coordinate value equal to some large number. The user changes to a logarithmic scale, but forgets to change the minimum coordinate value of the scale from 0.0 to some positive non-zero number. The coordinate conversion routines will halt the first time the $\log(0.0)$ is in a calculation. The same is also true if the minimum coordinate value is any negative number. This software always checks for this condition and changes the minimum coordinate value using the following criteria. If the minimum coordinate value for a logarithmic scale is less than or equal to 0.0 it is assumed that the user made an error and coordinate value is set to 1.0. If the minimum coordinate value is greater than 0.0 but less than the value of `MIN_LOG_VALUE`, the minimum coordinate value is set to `MIN_LOG_VALUE`.

Positioning the Plot Area Volume in Graph Area Volume

The **WorkingCoordinates** class has a group of methods - **SetGraphBorderFrame**, **SetGraphBorderDiagonal**, and **SetGraphBorderInsets** - that position the plot area volume of the chart in the graph viewport. Since the coordinate system scaling classes are subclasses of **WorkingCoordinates**, these methods are part of those classes. These methods are redundant and only one need be called. The default position of the plot area

110 Scaling and Coordinate Systems

volume in the graph view port is at $x = 0.2$, $y = 0.2$, $z = 0.2$, width = 0.6, height = 0.6, depth = 0.6 specified using graph normalized coordinates.

This method initializes the position and size of the plot area volume inside the graph area volume, specified using a rectangle to specify graph normalized coordinates.

SetGraphBorder method

```
[Visual Basic]
Overloads Public Sub SetGraphBorderFrame( _
    ByVal border As Rectangle3D _
)

[C#]
public void SetGraphBorderFrame(
    Rectangle3D border
);
```

border Specifies the rectangle defining the plot area volume border.

This method initializes the position and size of the plot area volume inside the graph area volume, specified using graph normalized position and size values.

```
[Visual Basic]
Overloads Public Sub SetGraphBorderFrame( _
    ByVal rLeft As Double, _
    ByVal rTop As Double, _
    ByVal rFront As Double, _
    ByVal width As Double, _
    ByVal height As Double, _
    ByVal depth As Double _
)

Overloads Public Sub SetGraphBorderFrame( _
    ByVal rLeft As Double, _
    ByVal rTop As Double, _
    ByVal width As Double, _
    ByVal height As Double _
)

[C#]
public void SetGraphBorderFrame(
    double rLeft,
    double rTop,
    double rFront,
    double width,
    double height,
    double depth
);

public void SetGraphBorderFrame(
    double rLeft,
```

```

    double rTop,
    double width,
    double height,
);

```

where

<i>rLeft</i>	The left x-position of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>rTop</i>	The top y-position of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>rFront</i>	The front z-position of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>width</i>	The width of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>height</i>	The height of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>depth</i>	The depth of the plot area volume inside the graph area volume specified using graph normalized coordinates.

This method initializes the size and position of the plot area volume inside the graph area volume, specified using graph normalized values for the opposite corners of the region.

[Visual Basic]

```

Overloads Public Sub SetGraphBorderDiagonal( _
    ByVal rLeft As Double, _
    ByVal rTop As Double, _
    ByVal rFront As Double, _
    ByVal rRight As Double, _
    ByVal rBottom As Double, _
    ByVal rRear As Double _
)

```

```

Public Sub SetGraphBorderDiagonal( _
    ByVal rLeft As Double, _
    ByVal rTop As Double, _
    ByVal rRight As Double, _
    ByVal rBottom As Double _
)

```

[C#]

```

public void SetGraphBorderDiagonal(

```

112 Scaling and Coordinate Systems

```
double rLeft,  
double rTop,  
double rRight,  
double rBottom  
);  
  
public void SetGraphBorderDiagonal(  
double rLeft,  
double rTop,  
double rFront,  
double rRight,  
double rBottom,  
double rRear  
);
```

<i>rLeft</i>	The left x-position of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>rTop</i>	The top y-position of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>rFront</i>	The front z-position of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>rRight</i>	The right x-position of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>rBottom</i>	The bottom y-position of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>rRear</i>	The back z-position of the plot area volume inside the graph area volume specified using graph normalized coordinates.

This method initializes the insets of the plot area volume inside the graph area volume, specified using graph normalized values.

```
[Visual Basic]  
Public Sub SetGraphBorderInsets( _  
    ByVal rLeft As Double, _  
    ByVal rTop As Double, _  
    ByVal rRight As Double, _  
    ByVal rBottom As Double _  
)
```

```
Public Sub SetGraphBorderInsets( _
    ByVal rLeft As Double, _
    ByVal rTop As Double, _
    ByVal rFront As Double, _
    ByVal rRight As Double, _
    ByVal rBottom As Double, _
    ByVal rRear As Double _
)
```

```
[C#]
public void SetGraphBorderInsets(
    double rLeft,
    double rTop,
    double rRight,
    double rBottom
);

public void SetGraphBorderInsets(
    double rLeft,
    double rTop,
    double rFront,
    double rRight,
    double rBottom,
    double rRear
);
```

<i>rLeft</i>	The left inset of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>rTop</i>	The top inset of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>rTop</i>	The front inset of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>rRight</i>	The right inset of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>rBottom</i>	The bottom inset of the plot area volume inside the graph area volume specified using graph normalized coordinates.
<i>rRear</i>	The rear inset of the plot area volume inside the graph area volume specified using graph normalized coordinates.

The following examples all position the plot area volume of the chart in the upper right quadrant of the graph viewport.

```
[C#]
```

114 *Scaling and Coordinate Systems*

```
CartesianCoordinates simpleScale;
// Assume the default value of 0.0 to 1.0 for z-dimension
simpleScale = new CartesianCoordinates(xMin, yMin, xMax, yMax);

// Use ONE of the example below
// Example #1
simpleScale.SetGraphBorderFrame(new Rectangle3D(0.5, 0.0, 0.5, 0.5));

// Example #2
simpleScale.SetGraphBorderFrame(0.5, 0.0, 0.5, 0.5);

// Example #3
simpleScale.SetGraphBorderDiagonal (0.5, 0.0, 1.0, 0.5);

// Example #4
simpleScale.SetGraphBorderInsets (0.5, 0.0, 0.0, 0.5);
```

[Visual Basic]

```
Dim xMin As Double = -5
Dim xMax As Double = 15
Dim yMin As Double = 0
Dim yMax As Double = 105

Dim simpleScale As CartesianCoordinates
simpleScale = New CartesianCoordinates(xMin, yMin, xMax, yMax)

` Use ONE of the example below
` Example #1
simpleScale.SetGraphBorderFrame(new Rectangle3D(0.5, 0.0, 0.5, 0.5))

` Example #2
simpleScale.SetGraphBorderFrame(0.5, 0.0, 0.5, 0.5)

` Example #3
simpleScale.SetGraphBorderDiagonal (0.5, 0.0, 1.0, 0.5)

` Example #4
simpleScale.SetGraphBorderInsets (0.5, 0.0, 0.0, 0.5)
```

Rotating the Chart in the Graph Area Volume

The **WorkingCoordinate** base class for all coordinate system classes includes routines for rotating the chart within the chart area volume. The default rotation of the chart is no rotation (x-rotation = y-rotation = z-rotation = 0) where the XY plane is parallel to the screen, and the z-axis is perpendicular to the screen. This is the standard 2D representation of data found in our **QCChart2D** software, and others that do not support 3D graphics. If you want your charts plotted with a 3D look and feel, you must explicitly rotate the chart in your program, using one the coordinate systems

AbsRotateCoordinateSystem and/or **RelRotateCoordinateSystem** methods. The default origin of the rotations, i.e. the point around which the rotations are viewed to take place, is the 3D point (0.5, 0.5, 0.5), the centroid of the graph area volume in normalized coordinates. This can be changed using the **AbsTranslateCoordinateSystem** method.

AbsRotateCoordinateSystem

Set absolute the rotational state of the coordinate system.

```
[Visual Basic]
Overloads Public Sub AbsRotateCoordinateSystem( _
    ByVal rotate As Point3D _
)
```

```
[C#]
public void AbsRotateCoordinateSystem(
    Point3D rotate
);
```

rotate Rotate the coordinate system by this amount in absolute degrees.

Rotating a chart to an absolute position of 10 degrees about the x-axis and 20 degrees about the y-axis. Extracted from the Bargraphs.SimpleBars example program.

```
// Auto-scale a coordinate system for data
pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(datasetarray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
chartVu.SetFractionalZViewportDepth(0.6);
pTransform1.SetGraphBorderDiagonal(0.15, 0.2, .8, 0.7) ;
// set the rotational characteristics of the chart
pTransform1.AbsRotateCoordinateSystem(new Point3D(10,20,0));
```

RelRotateCoordinateSystem

Incrementally rotate the state of the coordinate system.

```
[Visual Basic]
Overloads Public Sub RelRotateCoordinateSystem( _
    ByVal rotate As Point3D _
)

[C#]
public void RelRotateCoordinateSystem(
    Point3D rotate
);
```

Parameters

rotate

Rotate the coordinate system by this amount in relative degrees.

RelRotateCoordinateSystem

Incrementally rotate one axis of the coordinate system.

```
[Visual Basic]
Overloads Public Sub RelRotateCoordinateSystem( _
    ByVal degree As Double, _
    ByVal iAxis As Integer _
)

[C#]
public void RelRotateCoordinateSystem(
    double degree,
    int iAxis
);
```

Parameters

degree

Rotate the coordinate system by this amount in relative degrees.

iAxis

Rotate about this axis (X_AXIS, Y_AXIS or Z_AXIS).

AbsTranslateCoordinateSystem

Set the absolute translational state of the coordinate system.

```
[Visual Basic]
Public Sub AbsTranslateCoordinateSystem( _
    ByVal translate As Point3D _
)

[C#]
public void AbsTranslateCoordinateSystem(
    Point3D translate
);
```

Parameters

translate

Translate the coordinate system by this amount, using normalized (0.0 to 1.0) coordinates.

Linear and Logarithmic Coordinate Scaling

Class **CartesianCoordinates**

PhysicalCoordinates

|
+-- **CartesianCoordinates**

The **CartesianCoordinates** class scales the chart plot area volume for a physical coordinate system that uses linear and/or logarithmic scaling.

There are three main ways to scale the plot area volume:

- Scale the minimum and maximum x- and y- values explicitly
- Use an auto-scale method that calculates appropriate minimum and maximum x- and y-values based on the x- and y-values in one or more datasets
- Use a combination of the first two methods. It is useful to be able to run an auto-scale function, and then change the minimum or maximum value of one or more coordinate endpoints.

Linear Coordinate Scaling

The default coordinate system for the **CartesianCoordinates** class is linear for x, y and z. If you already know the range for x, y and z for the plot area volume, you can scale the plot area volume explicitly. The example below uses a **CartesianCoordinates** constructor to initialize the coordinates to the proper values.

CartesianCoordinates constructor with explicit scaling, using the default 0.0 to 1.0 z-dimension scale

[C#]

```
double xMin = -5;
double xMax = 15;
double yMin = 0;
double yMax = 105;

CartesianCoordinates simpleScale;
simpleScale = new CartesianCoordinates(xMin, yMin, xMax, yMax);
```

[Visual Basic]

```
Dim xMin As Double = -5
Dim xMax As Double = 15
Dim yMin As Double = 0
Dim yMax As Double = 105

Dim simpleScale As CartesianCoordinates
simpleScale = New CartesianCoordinates(xMin, yMin, xMax, yMax)
```

CartesianCoordinates constructor with explicit 3D scaling for the x, y and z-dimension

[C#]

```
double xMin = -5;
double xMax = 15;
double yMin = 0;
double yMax = 105;
double zMin = 0;
double zMax = 10;

CartesianCoordinates simpleScale;
simpleScale = new CartesianCoordinates(xMin, yMin, zMin, xMax, yMax, zMax);
```

[Visual Basic]

```
Dim xMin As Double = -5
Dim xMax As Double = 15
Dim yMin As Double = 0
Dim yMax As Double = 105
Dim zMin As Double = 0
Dim zMax As Double = 10

Dim simpleScale As CartesianCoordinates
simpleScale = New CartesianCoordinates(xMin, yMin, zMin, xMax, yMax, zMax)
```

Another technique uses the default constructor and scales the coordinates using the **CartesianCoordinates.SetCoordinateBounds** method.

Example of explicit scaling of a CartesianCoordinates object using the CartesianCoordinates.SetCoordinateBounds method, using the default 0.0 to 1.0 z-dimension scale

[C#]

```
double xmin = -5;
double xmax = 15;
double ymin = 0;
double ymax = 105;
CartesianCoordinates simpleScale = new CartesianCoordinates();

simpleScale.SetCoordinateBounds(xmin, ymin, xmax, ymax);
```

[Visual Basic]

```
Dim xmin As Double = -5
Dim xmax As Double = 15
Dim ymin As Double = 0
Dim ymax As Double = 105

Dim simpleScale As CartesianCoordinates = New CartesianCoordinates()
simpleScale.SetCoordinateBounds(xmin, ymin, xmax, ymax)
```

Example of explicit scaling of a CartesianCoordinates object using the CartesianCoordinates.SetCoordinateBounds method, with explicit 3D scaling for the x, y and z-dimension

[C#]

```
double xmin = -5;
double xmax = 15;
double ymin = 0;
double ymax = 105;
double zmin = 0;
double zmax = 10;
CartesianCoordinates simpleScale = new CartesianCoordinates();

simpleScale.SetCoordinateBounds(xmin, ymin, zmin, xmax, ymax, zmax);
```

[Visual Basic]

```
Dim xmin As Double = -5
Dim xmax As Double = 15
```

120 *Scaling and Coordinate Systems*

```
Dim yMin As Double = 0
Dim yMax As Double = 105
Dim zMin As Double = 0
Dim zMax As Double = 10

Dim simpleScale As CartesianCoordinates = New CartesianCoordinates()
simpleScale.SetCoordinateBounds(xMin, yMin, zMin, xMax, yMax, zMax)
```

It is possible to scale the bounds of the coordinate system based on the data values in a dataset. There are constructors and methods that take a single dataset and others that take an array of datasets.

Example of auto-scaling a CartesianCoordinates object using a single dataset

[C#]

```
double [] xData = {1,2,3,4,5,6,7,8,9,10};
double [] yData = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9};
double zvalue = 0.0;
double depthvalue 1.0;
SimpleDataset dataset =
    new SimpleDataset("Sales", xData, yData, zvalue, depthvalue);
CartesianCoordinates simpleScale = new CartesianCoordinates();
simpleScale.AutoScale(dataset);
```

[Visual Basic]

```
Dim xData() As Double = {1,2,3,4,5,6,7,8,9,10}
Dim yData() As Double = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9}
Dim zvalue As Double = 0.0
Dim depthvalue As Double = 1.0

Dim dataset As SimpleDataset = _
    New SimpleDataset("Sales", xData, yData, svalue, depthvalue)
Dim simpleScale As CartesianCoordinates = New CartesianCoordinates()
simpleScale.AutoScale(dataset)
```

You can control the “tightness” of the auto-scale values about the dataset values using other versions of the **CartesianCoordinates.AutoScale** method that take rounding mode parameters.

Example of auto-scaling a CartesianCoordinates object using a single dataset and explicit rounding mode parameters

```
simpleScale.AutoScale(dataset, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
```

You can auto-scale the bounds of the coordinate system using a dataset, and then explicitly modify the range the auto-scale selected. There are methods that set the minimum and maximum values of the x- and y-scales. This way you can use the auto-scale methods for the values of one scale (the y-scale in the example below), but explicitly set the values for the other scale (the x-scale in the example below).

Example of modifying the minimum and maximum values selected by an auto-scale method.

[C#]

```
double [] xData = {2,3,4,5,6,7,8,9};
double [] yData = { 22, 33, 44, 55, 46, 33, 25, 14};
double zvalue = 0.5;
double depthvalue = 0.1;

SimpleDataset dataset =
    new SimpleDataset("Sales", xData, yData, zvalue, depthvalue);
CartesianCoordinates simpleScale = new CartesianCoordinates();
simpleScale.AutoScale(dataset);
simpleScale.SetScaleStopX(10);
simpleScale.SetScaleStartX(1.0);
```

[Visual Basic]

```
Dim xData() As Double = {1,2,3,4,5,6,7,8,9,10}
Dim yData() As Double = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9}
Dim zvalue As Double = 0.5
Dim depthvalue As Double = 0.1

Dim dataset As SimpleDataset = _
    New SimpleDataset("Sales", xData, yData, zvalue, depthvalue)
Dim simpleScale As CartesianCoordinates = new CartesianCoordinates()
simpleScale.AutoScale(dataset)
simpleScale.SetScaleStartX(1.0)
simpleScale.SetScaleStopX(10.0)
```

The auto-scale methods that use an array of datasets to determine the proper range are very similar.

Example of auto-scaling a `CartesianCoordinates` object using the multiple datasets

[C#]

```
double [] xData1 = {1,2,3,4,5,6,7,8,9,10};
double [] yData1 = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9};
double [] xData2 = {10,9,8,7,6,5,4,3,2,1};
double [] yData2 = {20, 12, 43, 54, 15, 26, 63, 25, 24, 19};
double [] xData3 = {5,6,7,6,5,4,5,6,7,8};
double [] yData3 = {30, 52, 13, 64, 25, 76, 13, 35, 24, 19};
double zvalue = 0.1;
double depthvalue = 0.2;

SimpleDataset dataset1 =
    new SimpleDataset("Sales1",xData1,yData1, zvalue, depthvalue);
double zvalue = 0.4;
SimpleDataset dataset2 =
    new SimpleDataset("Sales2",xData2,yData2, zvalue, depthvalue);
double zvalue = 0.7;
SimpleDataset dataset3 =
    new SimpleDataset("Sales3",xData3,yData3, zvalue, depthvalue);
SimpleDataset [] datasetsArray = new SimpleDatasets[3];
datasetsArray[0] = dataset1;
datasetsArray[1] = dataset2;
datasetsArray[2] = dataset3;
CartesianCoordinates simpleScale = new CartesianCoordinates();
simpleScale.AutoScale(datasetsArray);
```

[Visual Basic]

```
Dim xData1() As Double = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Dim yData1() As Double = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9}
Dim xData2() As Double = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}
Dim yData2() As Double = {20, 12, 43, 54, 15, 26, 63, 25, 24, 19}
Dim xData3() As Double = {5, 6, 7, 6, 5, 4, 5, 6, 7, 8}
Dim yData3() As Double = {30, 52, 13, 64, 25, 76, 13, 35, 24, 19}
Dim zvalue As Double = 0.1
Dim depthvalue As Double = 0.2
```

```

Dim dataset1 As SimpleDataset = _
    New SimpleDataset("Sales1", xData1, yData1, zvalue, depthvalue)
zvalue = 0.4
Dim dataset2 As SimpleDataset = _
    New SimpleDataset("Sales2", xData2, yData2, zvalue, depthvalue)
zvalue = 0.7

Dim dataset3 As SimpleDataset = _
    New SimpleDataset("Sales3", xData3, yData3, zvalue, depthvalue)
Dim datasetsArray(2) As SimpleDataset
datasetsArray(0) = dataset1
datasetsArray(1) = dataset2
datasetsArray(2) = dataset3
Dim simpleScale As CartesianCoordinates = New CartesianCoordinates()
simpleScale.AutoScale(datasetsArray)

```

There is a version of the multiple dataset auto-scale routine that also specifies rounding mode parameters.

```
simpleScale.AutoScale(datasetsArray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
```

Logarithmic Coordinate Scaling

The previous examples assume that both the x- and y- scales are linear. If the x and/or y scale are to be logarithmic, then use the **CartesianCoordinates** constructor that has scale mode parameters.

Example of explicit scaling of three different logarithmic CartesianCoordinates objects

[C#]

```

double xmin = 1;
double xmax = 1000;
double ymin = 0.2;
double ymax = 2000;

CartesianCoordinates logYScale =
    new CartesianCoordinates(ChartObj.LINEAR_SCALE, ChartObj.LOG_SCALE);
logYScale.SetCoordinateBounds(xmin, ymin, xmax, ymax );

CartesianCoordinates logXScale =

```

124 *Scaling and Coordinate Systems*

```
        new CartesianCoordinates(ChartObj.LOG_SCALE, ChartObj.LINEAR_SCALE);  
logXScale.SetCoordinateBounds(xMin, yMin, xMax, yMax);
```

```
CartesianCoordinates logXLogYScale =  
        new CartesianCoordinates(ChartObj.LOG_SCALE, ChartObj.LOG_SCALE);  
logXLogYScale.SetCoordinateBounds(xMin, yMin, xMax, yMax);
```

[Visual Basic]

```
Dim xMin As Double = 1  
Dim xMax As Double = 1000  
Dim yMin As Double = 0.2  
Dim yMax As Double = 2000  
Dim logYScale As CartesianCoordinates = _  
        New CartesianCoordinates(ChartObj.LINEAR_SCALE, ChartObj.LOG_SCALE)  
logYScale.SetCoordinateBounds(xMin, yMin, xMax, yMax)  
  
Dim logXScale As CartesianCoordinates = _  
        New CartesianCoordinates(ChartObj.LOG_SCALE, ChartObj.LINEAR_SCALE)  
logXScale.SetCoordinateBounds(xMin, yMin, xMax, yMax)  
  
Dim logXLogYScale As CartesianCoordinates = _  
        New CartesianCoordinates(ChartObj.LOG_SCALE, ChartObj.LOG_SCALE)  
logXLogYScale.SetCoordinateBounds(xMin, yMin, xMax, yMax)
```

Note: When you explicitly scale the minimum and maximum values for a scale set to logarithmic coordinates, make sure you use valid values, i.e. non-negative values greater than 0.0.

The auto-scale routines work for both linear and logarithmic scales. If you use the auto-scale methods, it is important that you call the auto-scale methods **after** you establish if a scale is linear or logarithmic. This is because the auto-scale routines will allow zero and negative values for the minimum and maximum of a linear scale, but not a logarithmic scale. If you call the auto-scale routines first, while the scales are set to the default linear scale mode, and change one or both of the scales to logarithmic mode, you can introduce invalid negative and zero values into the logarithmic coordinate system.

Example of auto-scaling a CartesianCoordinates object that has a logarithmic y-scale, using a single dataset

[C#]


```

double [] xData = {2,3,4,5,6,7,8,9};
double [] yData = { 2, 33, 440, 5554, 46123, 332322, 5435641, 64567551};

SimpleDataset dataset = new SimpleDataset("Sales", xData, yData);
CartesianCoordinates simpleScale =
    new CartesianCoordinates(ChartObj.LINEAR_SCALE, ChartObj.LOG_SCALE);
simpleScale.AutoScale(dataset);
simpleScale.SetScaleStopX(10);
simpleScale.SetScaleStartX(1.0);

```

[Visual Basic]

```

Dim xData() As Double = {2,3,4,5,6,7,8,9}
Dim yData() As Double = { 2, 33, 440, 5554, 46123, 332322, 5435641, 64567551}

Dim dataset As SimpleDataset = New SimpleDataset("Sales", xData, yData)
Dim simpleScale As CartesianCoordinates = _
    New CartesianCoordinates(ChartObj.LINEAR_SCALE, ChartObj.LOG_SCALE)
simpleScale.AutoScale(dataset)
simpleScale.SetScaleStopX(10)
simpleScale.SetScaleStartX(1.0)

```

Coordinate Systems using times and dates

Many charting applications use data where the x-coordinate values are in the form of time and date records. The creation of a powerful yet flexible time and date physical coordinate system for scaling charts is a major challenge. Important design goals include:

- The programmer scales the graph using objects of the **ChartCalendar** class.
- The scale should support a continuous range of date/time scaling from milliseconds to hundreds of years.
- The scale should have a 5/7 day week option.
- A day does not have to be 24 hours long; instead, it can have a specific range, for example: 8:30 AM to 4:00 PM.
- Time and date axes, used to delineate a date/time scale, must be able to display and label tick marks for days, weeks, months and years, taking into account the non-uniformity of years, months, weeks and days in the Gregorian calendar.

The ChartCalendar Class

The **ChartCalendar** class represents time and date information in the format used by the majority of the world. It is a universal time class, capable of representing time with a resolution of milliseconds and a dynamic range of hundreds of years. It contains time and date fields that specify an exact moment in time, i.e. November 7, 2000 20:43:22.554. Since the **ChartCalendar** class represents both time of day and calendar dates, is not necessary to use separate time of day and date classes to manage data collected every few seconds, yet spans a day or more. An example of this type of date/time range is a graph of experimental data, sampled every 15 seconds, starting on June 12, 2001 11:43:00 PM and continuing until June 15, 2001 1:03:45 AM.

The major application for date/time chart scaling on the Internet is the display of financial market data. Multiply the number of on-line investors using the charting tools of on-line brokerage firms and financial web sites by the number of individual stocks, bonds, options and futures contracts that they are able to chart and it becomes apparent that the number of potential charts is almost infinite. Many stocks have more than 50 years of hourly historical information associated with them. A chart comparing the relative performance of General Electric, IBM, Dupont, Ford and Kodak, compared to the Dow Jones Industrial average, since the dawn of the computer age in 1950, will quickly highlight the above average stock market gains that can be made by investing in the right stocks. A user may start by looking at a fifty-year window on a stock, then through successive zoom operations narrow the window to two years, one month, one day, one hour and perhaps even one minute.

5 vs. 7 Day Work Weeks

Historically, western cultures use a five-day workweek. Much of the data suitable for charting includes data for only the workdays of Monday through Friday, excluding weekends. Data associated with financial markets is the most common. Stocks trade on Monday through Friday, excluding weekends and holidays. In the display of financial market information, it is not useful, and in many cases misleading, to scale a graph, using a seven-day work week, and then only plot data using five of the seven days. The gaps left in the chart waste valuable chart space. For line plots, if a line is drawn from the last data point on Friday to the first data point on Monday, the result gives a visual impression that the trend from Friday to Monday lasted two days (Friday midnight to Sunday midnight), when in fact it may have only lasted minutes. The **QCChart3D for .Net** date/time scaling, axes and auto-axes classes support dropping weekends, as an option, from the scale. One minute after Friday midnight is 12:01 AM Monday morning.

Non-24 Hour Days

Similarly, a full day of time stamped data may not fill an entire 24 hour day. Financial markets have specific trading hours. For example, the NYSE is open from 9:30 AM EST to 4:00 PM EST. Stocks traded on the NYSE will have a time stamp in this time range. Plotting NYSE stock data for several days, using data points sampled every fifteen

minutes, will result in large gaps in a traditional chart, where 2/3 of the day stock trading is inactive. It is possible to treat the unused portion of the day in a manner analogous to weekends. It is possible to eliminate unused hours and minutes of a day from the chart coordinate system. The **QCChart3D for .Net** coordinate system allows the programmer to specify a starting and ending time for a days worth of data. In the NYSE stock market example the starting time is 9:30 AM EST and the ending time is 4:00 PM EST. Any data outside of this range is invalid and not plotted. In terms of the resulting chart, one minute after 4:30 PM is 9:31 AM. Combine the starting and ending time parameters, with the option of deleting weekends from consideration, and one minute after 4:30 PM Friday is 9:31 AM Monday.

Non-Uniformity of Date/Time Tick Marks

There even more complications associated with date/time scales. The axis that delineates a date/time scale must take into account the non-uniform nature of date/time tick marks and tick mark labels. A month has a variable number of days. When months are used as the major tick mark interval, and days are the minor tick mark interval, the software must be capable of plotting 28, 29, 30 or 31 minor tick marks (days) for every major tick mark (months), depending on the month. Another example is the use of months as the major tick mark interval, and weeks as the minor tick mark interval. Some months will have four minor tick marks (start of each new week) while others will have five. The software also needs to take into the variable number of days/year due to leap years. Date/time axis tick marks and labels become even more complicated when the 5-day/7-day option and the working hours/day options are used.

The **QCChart3D for .Net** library uses milliseconds as the underlying time base for all date/time coordinate system. When a scale is created using two **ChartCalendar** dates as end points, the software calculates the number of milliseconds seconds between the starting date and the ending date. If the coordinate system is based on a 5-day week, the milliseconds associated with the missing weekends are not counted. If the coordinate system does not use 24 hours a day, the milliseconds associated with the missing part of the day are not counted. A linear coordinate system is scaled using the range of calculated milliseconds. Data points plotted in this coordinate system have their date/time value converted to milliseconds seconds by subtracting the starting date of the scale from the data point date, making sure to exclude the seconds associated with weekends and fractional days, if necessary. The data point is plotted in the milliseconds based linear coordinate system. Since the **QCChart3D for .Net** library uses milliseconds as the underlying time base, the minimum allowable displayable range is one millisecond. For ranges smaller than one second, the programmer needs to convert the **ChartCalendar** values to seconds or milliseconds and use the **CartesianCoordinates** class to scale the chart. You loose the date/time formatting of the axis labels, but this should not matter if you are dealing in the sub millisecond realm.

Class TimeCoordinates**PhysicalCoordinates**

```

|
+-- TimeCoordinates

```

The **TimeCoordinates** class scales the chart plot area volume for physical coordinate systems that use date/time scaling. The basic techniques are essentially the same as those used with linear and logarithmic scaling; only the **TimeCoordinates** class uses **ChartCalendar** dates in place of numeric values for the x-axis scale values. The minimum and maximum values of the x-, y-, and z-scales can be set explicitly, set using one of the auto-scale methods, or set using a combination of the two.

The default coordinate system for the **TimeCoordinates** class is time for the x-scale, linear for the y-scale, and linear for the z-scale. The y- and z-scales can be logarithmic and you can set that mode using the explicit scale mode version of the **TimeCoordinates** constructor. If you already know the range for x, y and z for the plot area volume, you can scale the plot area volume explicitly. In the example below a **TimeCoordinates** constructor initializes the coordinates to the proper values.

TimeCoordinates constructor with explicit scaling

[C#]

```

ChartCalendar xMin = new ChartCalendar(1996, ChartObj.FEBRUARY,5);
ChartCalendar xMax = new ChartCalendar(2002, ChartObj.JANUARY,5);
double yMin = 0;
double yMax = 105;
double zMin = 0;
double zMax = 10;

TimeCoordinates simpleTimeScale;
simpleTimeScale = new TimeCoordinates(xMin, yMin, zMin, xMax, yMax, zMax);

```

[Visual Basic]

```

Dim xMin As ChartCalendar = New ChartCalendar(1996, ChartObj.FEBRUARY,5)
Dim xMax As ChartCalendar = New ChartCalendar(2002, ChartObj.JANUARY,5)
Dim yMin As Double = 0
Dim yMax As Double = 105
Dim zMin As Double = 0
Dim zMax As Double = 10

Dim simpleTimeScale As TimeCoordinates

```

```
simpleTimeScale = new TimeCoordinates(xMin, yMin, zMin, xMax, yMax, zMax)
```

Another technique uses the default constructor and scales the coordinates using the **TimeCoordinates.SetTimeCoordinateBounds** method.

Example of explicit scaling of a TimeCoordinates object using the TimeCoordinates.SetTimeCoordinateBounds method

[C#]

```
ChartCalendar xMin = new ChartCalendar(1996, ChartObj.FEBRUARY,5);
ChartCalendar xMax = new ChartCalendar(2002, ChartObj.JANUARY,5);
double yMin = 0;
double yMax = 105;
double zMin = 0;
double zMax = 10;

TimeCoordinates simpleTimeScale = new TimeCoordinates();

simpleTimeScale.SetTimeCoordinateBounds (xMin, yMin, zMin, xMax, yMax, zMax);
```

[Visual Basic]

```
Dim xMin As ChartCalendar = New ChartCalendar(1996, ChartObj.FEBRUARY,5)
Dim xMax As ChartCalendar = New ChartCalendar(2002, ChartObj.JANUARY,5)
Dim yMin As Double = 0
Dim yMax As Double = 105
Dim zMin As Double = 0
Dim zMax As Double = 10

Dim simpleTimeScale As TimeCoordinates = New TimeCoordinates()
simpleTimeScale.SetTimeCoordinateBounds (xMin, yMin, zMin, xMax, yMax, zMax)
```

It is possible to scale the bounds of the coordinate system based on the data values in a time-based dataset, **TimeSimpleDataset** and **TimeGroupDataset**. There are constructors and methods that take a single dataset and others that take an array of datasets.

Example of auto-scaling a TimeCoordinates object using a single dataset

130 Scaling and Coordinate Systems

[C#]

```
ChartCalendar [] xData = { new ChartCalendar(1996, ChartObj.FEBRUARY, 5),
                           new ChartCalendar(1996, ChartObj.MARCH, 5),
                           new ChartCalendar(1996, ChartObj.APRIL, 5),
                           new ChartCalendar(1996, ChartObj.MAY, 5),
                           new ChartCalendar(1996, ChartObj.JUNE, 5),
                           new ChartCalendar(1996, ChartObj.JULY, 5),
                           new ChartCalendar(1996, ChartObj.AUGUST, 5),
                           new ChartCalendar(1996, ChartObj.SEPTEMBER, 5),
                           new ChartCalendar(1996, ChartObj.OCTOBER, 5),
                           new ChartCalendar(1996, ChartObj.NOVEMBER, 5)};

double [] yData = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9};
double zvalue = 0.0
double depthvalue = 1.0
TimeSimpleDataset dataset =
    new TimeSimpleDataset("Sales", xData, yData, zvalue, depthvalue);
TimeCoordinates simpleTimeScale = new TimeCoordinates();
simpleTimeScale.AutoScale(dataset);
```

[Visual Basic]

```
Dim xData() As ChartCalendar = { New ChartCalendar(1996, ChartObj.FEBRUARY, 5), _
                                New ChartCalendar(1996, ChartObj.MARCH, 5), _
                                New ChartCalendar(1996, ChartObj.APRIL, 5), _
                                New ChartCalendar(1996, ChartObj.MAY, 5), _
                                New ChartCalendar(1996, ChartObj.JUNE, 5), _
                                New ChartCalendar(1996, ChartObj.JULY, 5), _
                                New ChartCalendar(1996, ChartObj.AUGUST, 5), _
                                New ChartCalendar(1996, ChartObj.SEPTEMBER, 5), _
                                New ChartCalendar(1996, ChartObj.OCTOBER, 5), _
                                New ChartCalendar(1996, ChartObj.NOVEMBER, 5)}

Dim yData() As Double = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9}
Dim zvalue As Double = 0.0;
Dim depthvalue As Double = 1.0;

Dim dataset As TimeSimpleDataset = _
    New TimeSimpleDataset("Sales", xData, yData, zvalue, depthvalue)
Dim simpleTimeScale As TimeCoordinates = New TimeCoordinates()
simpleTimeScale.AutoScale(dataset)
```

You can control the “tightness” of the auto-scale values about the dataset values using other versions of the **TimeCoordinates.AutoScale** method that take rounding mode parameters.

Example of auto-scaling a TimeCoordinates object using a single dataset and explicit rounding mode parameters

```
simpleTimeScale.AutoScale(dataset, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
```

You can auto-scale the coordinate bounds using a dataset, and then explicitly modify the range the auto-scale selected. There are methods for setting the minimum and maximum values of the x-, y- and z-scales. This way you can use the auto-scale methods for the values of one scale (the y-scale in the example below), but explicitly set the values for the other scale (the x-scale in the example below).

Example of modifying the minimum and maximum values selected by an auto-scale method

[C#]

```
ChartCalendar [] xData = { new ChartCalendar(1996, ChartObj.FEBRUARY, 5),
                           new ChartCalendar(1996, ChartObj.MARCH, 5),
                           new ChartCalendar(1996, ChartObj.APRIL, 5),
                           new ChartCalendar(1996, ChartObj.MAY, 5),
                           new ChartCalendar(1996, ChartObj.JUNE, 5),
                           new ChartCalendar(1996, ChartObj.JULY, 5),
                           new ChartCalendar(1996, ChartObj.AUGUST, 5),
                           new ChartCalendar(1996, ChartObj.SEPTEMBER, 5),
                           new ChartCalendar(1996, ChartObj.OCTOBER, 5),
                           new ChartCalendar(1996, ChartObj.NOVEMBER, 5) };

double [] yData = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9};
double zvalue = 0.0;
double depthvalue = 1.0;

TimeSimpleDataset dataset =
    new TimeSimpleDataset("Sales", xData, yData, zvalue, depthvalue);
TimeCoordinates simpleTimeScale = new TimeCoordinates();
simpleTimeScale.AutoScale(dataset);
simpleTimeScale.SetTimeScaleStart(new ChartCalendar(1996, ChartObj.JANUARY, 5));
simpleTimeScale.SetTimeScaleStop(new ChartCalendar(1997, ChartObj.JANUARY, 5));
```

132 *Scaling and Coordinate Systems*

[Visual Basic]

```
Dim xData() As ChartCalendar = { New ChartCalendar(1996, ChartObj.FEBRUARY, 5), _  
    New ChartCalendar(1996, ChartObj.MARCH, 5), _  
    New ChartCalendar(1996, ChartObj.APRIL, 5), _  
    New ChartCalendar(1996, ChartObj.MAY, 5), _  
    New ChartCalendar(1996, ChartObj.JUNE, 5), _  
    New ChartCalendar(1996, ChartObj.JULY, 5), _  
    New ChartCalendar(1996, ChartObj.AUGUST, 5), _  
    New ChartCalendar(1996, ChartObj.SEPTEMBER, 5), _  
    New ChartCalendar(1996, ChartObj.OCTOBER, 5), _  
    New ChartCalendar(1996, ChartObj.NOVEMBER, 5) }  
  
Dim yData() As Double = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9}  
Dim zvalue As Double = 0.0  
Dim depthvalue As Double = 1.0  
  
Dim dataset As TimeSimpleDataset = _  
    New TimeSimpleDataset("Sales", xData, yData, zvalue, depthvalue)  
Dim simpleTimeScale As TimeCoordinates = New TimeCoordinates()  
simpleTimeScale.AutoScale(dataset)  
simpleTimeScale.SetTimeScaleStart(new ChartCalendar(1996, ChartObj.JANUARY, 5))  
simpleTimeScale.SetTimeScaleStop(new ChartCalendar(1997, ChartObj.JANUARY, 5))
```

The auto-scale methods that use an array of datasets to determine the proper range are very similar.

Example of auto-scaling a TimeCoordinates object using the multiple datasets

[C#]

```
ChartCalendar [] xData = { new ChartCalendar(1996, ChartObj.FEBRUARY, 5),  
    new ChartCalendar(1996, ChartObj.MARCH, 5),  
    new ChartCalendar(1996, ChartObj.APRIL, 5),  
    new ChartCalendar(1996, ChartObj.MAY, 5),  
    new ChartCalendar(1996, ChartObj.JUNE, 5),  
    new ChartCalendar(1996, ChartObj.JULY, 5),  
    new ChartCalendar(1996, ChartObj.AUGUST, 5),  
    new ChartCalendar(1996, ChartObj.SEPTEMBER, 5),  
    new ChartCalendar(1996, ChartObj.OCTOBER, 5),  
    new ChartCalendar(1996, ChartObj.NOVEMBER, 5) };
```



```

double [] yData1 = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9};
double [] yData2 = {20, 12, 43, 54, 15, 26, 63, 25, 24, 19};
double [] yData3 = {30, 52, 13, 64, 25, 76, 13, 35, 24, 19};

double zvalue1 = 0.0;
double zvalue2 = 0.3;
double zvalue3 = 0.6;
double depthvalue = 0.2;

// All of the datasets reference the same xData array of ChartCalendar
// dates, though this does not have to be the case.
TimeSimpleDataset dataset1 =
    new TimeSimpleDataset("Sales1",xData,yData1, zvalue1, depthvalue);
TimeSimpleDataset dataset2 =
    new TimeSimpleDataset("Sales2",xData,yData2, zvalue2, depthvalue);
TimeSimpleDataset dataset3 =
    new TimeSimpleDataset("Sales3",xData,yData3, zvalue3, depthvalue);
TimeSimpleDataset [] datasetsArray = new TimeSimpleDataset[3];
datasetsArray[0] = dataset1;
datasetsArray[1] = dataset2;
datasetsArray[2] = dataset3;
TimeCoordinates simpleTimeScale = new TimeCoordinates();
simpleTimeScale.AutoScale(datasetsArray);

```

[Visual Basic]

```

Dim xData() As ChartCalendar = { New ChartCalendar(1996, ChartObj.FEBRUARY, 5), _
    New ChartCalendar(1996, ChartObj.MARCH, 5), _
    New ChartCalendar(1996, ChartObj.APRIL, 5), _
    New ChartCalendar(1996, ChartObj.MAY, 5), _
    New ChartCalendar(1996, ChartObj.JUNE, 5), _
    New ChartCalendar(1996, ChartObj.JULY, 5), _
    New ChartCalendar(1996, ChartObj.AUGUST, 5), _
    New ChartCalendar(1996, ChartObj.SEPTEMBER, 5), _
    New ChartCalendar(1996, ChartObj.OCTOBER, 5), _
    New ChartCalendar(1996, ChartObj.NOVEMBER, 5)}

Dim yData1() As Double = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9}
Dim yData2() As Double = {20, 12, 43, 54, 15, 26, 63, 25, 24, 19}
Dim yData3() As Double = {30, 52, 13, 64, 25, 76, 13, 35, 24, 19}
Dim zvalue1 As Double = 0.0
Dim zvalue2 As Double = 0.3
Dim zvalue3 As Double = 0.6

```

134 Scaling and Coordinate Systems

```
Dim depthvalue As Double = 1.0

` All of the datasets reference the same xData array of ChartCalendar
` dates, though this does not have to be the case.
Dim dataset1 As TimeSimpleDataset = _
    New TimeSimpleDataset("Sales1",xData,yData1, xvalue1, depthvalue)
Dim dataset2 As TimeSimpleDataset = _
    New TimeSimpleDataset("Sales2",xData,yData2 xvalue2, depthvalue)
Dim dataset3 As TimeSimpleDataset = _
    New TimeSimpleDataset("Sales3",xData,yData3 xvalue13 depthvalue)
Dim datasetsArray(2) As TimeSimpleDataset
datasetsArray(0) = dataset1
datasetsArray(1) = dataset2
datasetsArray(2) = dataset3
Dim simpleTimeScale As TimeCoordinates = New TimeCoordinates()
simpleTimeScale.AutoScale(datasetsArray)
```

There is a version of the multiple dataset auto-scale routine that also specifies rounding mode parameters.

```
simpleTimeScale.AutoScale(datasetsArray,ChartObj.AUTOAXES_FAR,
ChartObj.AUTOAXES_FAR);
```

The previous examples use the **TimeCoordinates** default 7 days/week, and 24-hours/day modes. Many of the methods in the software have a *weektype* parameter. The default value of this parameter is the constant **ChartObj.WEEK_7D**. If you want the coordinate system to ignore Saturdays and Sundays, use the constant **ChartObj.WEEK_5D** constant. Use the methods below to establish a 5-days/week coordinate system.

TimeCoordinates constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal dstart As ChartCalendar, _
    ByVal y1 As Double, _
    ByVal dstop As ChartCalendar, _
    ByVal y2 As Double, _
    ByVal nweektype As Integer _
)
Overloads Public Sub New( _
    ByVal dstart As ChartCalendar, _
    ByVal y1 As Double, _
    ByVal z1 As Double, _
    ByVal dstop As ChartCalendar, _
    ByVal y2 As Double, _
```

```

        ByVal z2 As Double _
    )

[C#]

public TimeCoordinates(
    ChartCalendar dstart,
    double y1,
    ChartCalendar dstop,
    double y2,
    int nweektype
);

public TimeCoordinates(
    ChartCalendar dstart,
    double y1,
    double z1,
    ChartCalendar dstop,
    double y2,
    double z2,
    int nweektype
);

```

SetTimeCoordinateBounds method

```

[Visual Basic]
Overloads Public Sub SetTimeCoordinateBounds( _
    ByVal dstart As ChartCalendar, _
    ByVal y1 As Double, _
    ByVal dstop As ChartCalendar, _
    ByVal y2 As Double, _
    ByVal nweektype As Integer _
)

Overloads Public Sub SetTimeCoordinateBounds( _
    ByVal dstart As ChartCalendar, _
    ByVal y1 As Double, _
    ByVal z1 As Double, _
    ByVal dstop As ChartCalendar, _
    ByVal y2 As Double, _
    ByVal z2 As Double _
)

[C#]

public void SetTimeCoordinateBounds(
    ChartCalendar dstart,
    double y1,
    ChartCalendar dstop,
    double y2,
    int nweektype
);

public void SetTimeCoordinateBounds(
    ChartCalendar dstart,
    double y1,
    double z1,
    ChartCalendar dstop,
    double y2,
    double z2,
    int nweektype
);

```

SetWeekType method

```
[Visual Basic]
Public Sub SetWeekType( _
    ByVal weektype As Integer _
)

[C#]
public void SetWeekType(
    int weektype
);
```

If you use the auto-scale routines, set the week type before you call the **TimeCoordinates.AutoScale** method because the auto-scale routines need to take into account the week type. For example:

[C#]

```
TimeSimpleDataset dataset = new TimeSimpleDataset("Sales", xData, yData);
TimeCoordinates simpleTimeScale = new TimeCoordinates();
simpleTimeScale.SetWeekType(ChartObj.WEEK_5D);
simpleTimeScale.AutoScale(dataset);
```

[Visual Basic]

```
Dim dataset As TimeSimpleDataset = New TimeSimpleDataset("Sales", xData, yData)
Dim simpleTimeScale As TimeCoordinates = New TimeCoordinates()
simpleTimeScale.SetWeekType(ChartObj.WEEK_5D)
simpleTimeScale.AutoScale(dataset)
```

In addition to the week type, the other major way to customize a **TimeCoordinates** coordinate system is not to use a 24-hour day. There are methods that set the starting and ending time-of-day. For example, if you are interested in plotting stock market data trading during the regular trading day of 9:30 AM to 4:00 PM, you can setup the coordinate system to only include these hours, and to treat any data outside of these hours as invalid and not to be plotted. A day can have only one continuous range. You are not able to define a day to with a valid range of 9:30 AM to 4:00 PM and 6:00 PM to 9:00 PM. Only one of these ranges is valid, or a combined range of 9:30 AM to 9:00 PM where the 4:00 PM to 6:00 PM time segment is included in the range.

TimeCoordinates constructor with time-of-day parameters

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal dstart As ChartCalendar, _
    ByVal starttime As Long, _
    ByVal y1 As Double, _
    ByVal z1 As Double, _
    ByVal dstop As ChartCalendar, _
```

```

        ByVal stoptime As Long, _
        ByVal y2 As Double, _
        ByVal z2 As Double, _
        ByVal nweektype As Integer _
    )

    Overloads Public Sub New( _
        ByVal dstart As ChartCalendar, _
        ByVal starttime As Long, _
        ByVal y1 As Double, _
        ByVal dstop As ChartCalendar, _
        ByVal stoptime As Long, _
        ByVal y2 As Double, _
        ByVal nweektype As Integer _
    )

    [C#]

    public TimeCoordinates(
        ChartCalendar dstart,
        long starttime,
        double y1,
        ChartCalendar dstop,
        long stoptime,
        double y2,
        int nweektype
    );

    public TimeCoordinates(
        ChartCalendar dstart,
        long starttime,
        double y1,
        double z1,
        ChartCalendar dstop,
        long stoptime,
        double y2,
        double z2,
        int nweektype
    );

```

TimeCoordinates constructor example for a 5 day/week and 9:30 AM to 4:00 PM time-of-day range

```

[C#]

long starttod = (9 * 60 + 30) * 60 * 1000; // msec cooresponding to 9:30 AM
long stoptod = 16 * 60 * 60 * 1000; // msec cooresponding to 4:00 PM
ChartCalendar dstart = new ChartCalendar(1996,ChartObj.FEBRUARY,5);
ChartCalendar dstop = new ChartCalendar(1997, ChartObj.JANUARY,5);
double y1 = 0.0;
double y2 = 55.0;
TimeCoordinates stockTimeScale;

stockTimeScale = new TimeCoordinates(dstart, starttod, y1,
                                     dstop, stoptod , y2, ChartObj.WEEK_5D);

```

[VB]

```

Dim starttod As Long = (9 * 60 + 30) * 60 * 1000 ` msec cooresponding to 9:30 AM
Dim stoptod As Long = 16 * 60 * 60 * 1000 ` msec cooresponding to 4:00 PM

```

138 Scaling and Coordinate Systems

```
Dim dstart As ChartCalendar = New ChartCalendar(1996,ChartObj.FEBRUARY,5)
Dim dstop As ChartCalendar = New ChartCalendar(1997, ChartObj.JANUARY,5)
Dim y1 As Double = 0.0
Dim y2 As Double = 55.0
Dim stockTimeScale As TimeCoordinates

stockTimeScale = New TimeCoordinates(dstart, starttod, y1, dstop, stoptod , y2, _
                                     ChartObj.WEEK_5D)
```

Another technique uses the default constructor and scales the coordinates using the **TimeCoordinates.SetTimeCoordinateBounds** method.

SetTimeCoordinateBounds Method

```
[Visual Basic]
Overloads Public Sub SetTimeCoordinateBounds( _
    ByVal dstart As ChartCalendar, _
    ByVal starttod As Long, _
    ByVal y1 As Double, _
    ByVal dstop As ChartCalendar, _
    ByVal stoptod As Long, _
    ByVal y2 As Double, _
    ByVal nweektype As Integer _
)

Overloads Public Sub SetTimeCoordinateBounds( _
    ByVal dstart As ChartCalendar, _
    ByVal y1 As Double, _
    ByVal z1 As Double, _
    ByVal dstop As ChartCalendar, _
    ByVal y2 As Double, _
    ByVal z2 As Double _
)

[C#]

public void SetTimeCoordinateBounds(
    ChartCalendar dstart,
    double y1,
    double z1,
    ChartCalendar dstop,
    double y2,
    double z2,
    int nweektype
);

public void SetTimeCoordinateBounds(
    ChartCalendar dstart,
    long starttod,
    double y1,
    ChartCalendar dstop,
    long stoptod,
    double y2,
    int nweektype
);
```

SetTimeCoordinateBounds example for a 5 day/week and 9:30 AM to 4:00 PM time-of-day range **SetWeekType** method

[C#]

```

long starttod = (9 * 60 + 30) * 60 * 1000; // msec cooresponding to 9:30 AM
long stoptod = 16 * 60 * 60 * 1000; // msec cooresponding to 4:00 PM
ChartCalendar dstart = new ChartCalendar(1996, ChartObj.FEBRUARY, 5);
ChartCalendar dstop = new ChartCalendar(1997, ChartObj.JANUARY, 5);
double y1 = 0.0;
double y2 = 55.0;
TimeCoordinates stockTimeScale;

stockTimeScale = new TimeCoordinates();
stockTimeScale.SetTimeCoordinateBounds(dstart, starttod, y1,
                                       dstop, stoptod, y2,
                                       ChartObj.WEEK_5D);

```

[Visual Basic]

```

Dim starttod As Long = (9 * 60 + 30) * 60 * 1000 ' msec cooresponding to 9:30 AM
Dim stoptod As Long = 16 * 60 * 60 * 1000 ' msec cooresponding to 4:00 PM
Dim dstart As ChartCalendar = New ChartCalendar(1996, ChartObj.FEBRUARY, 5)
Dim dstop As ChartCalendar = New ChartCalendar(1997, ChartObj.JANUARY, 5)
Dim y1 As Double = 0.0
Dim y2 As Double = 55.0

Dim stockTimeScale As TimeCoordinates = New TimeCoordinates()
stockTimeScale.SetTimeCoordinateBounds(dstart, starttod, y1, _
                                       dstop, stoptod, y2, ChartObj.WEEK_5D)

```

If you use the auto-scale routines, set the week type and the time-of-day range before you call the **TimeCoordinates.AutoScale** method because the auto-scale routines need to take into account the number of seconds per day and the week type. For example:

[C#]

```

// the tradingDay array is initialized with the stock trading dates
// the stockPrice array is initialized with stock price data
TimeSimpleDataset Dataset1 = new TimeSimpleDataset("First", tradingDay, stockPrice);

long startTime = (9 * 60 + 30) * 60 * 1000; // msec cooresponding to 9:30 AM
long stopTime = 16 * 60 * 60 * 1000; // msec cooresponding to 4:00 PM

TimeCoordinates stockTimeScale = new TimeCoordinates();
stockTimeScale.SetWeekType (ChartObj.WEEK_5D);
stockTimeScale.SetScaleStartTOD(startTime);

```

140 Scaling and Coordinate Systems

```
stockTimeScale.SetScaleStopTOD(stopTime);  
stockTimeScale.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
```

[Visual Basic]

```
` the tradingDay array is initialized with the stock trading dates  
` the stockPrice array is initialized with stock price data  
Dim Dataset1 As TimeSimpleDataset = _  
    New TimeSimpleDataset("First", tradingDay, stockPrice)  
  
Dim startTime As Long = (9 * 60 + 30) * 60 * 1000 `msecs cooresponding to 9:30 AM  
Dim stopTime As Long = 16 * 60 * 60 * 1000 ` msecs cooresponding to 4:00 PM  
  
Dim stockTimeScale As TimeCoordinates = new TimeCoordinates()  
stockTimeScale.SetWeekType (ChartObj.WEEK_5D)  
stockTimeScale.SetScaleStartTOD(startTime)  
stockTimeScale.SetScaleStopTOD(stopTime)  
stockTimeScale.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
```

Miscellaneous Coordinate System Topics

Inverted Coordinate Systems

Charts that use linear, logarithmic and time coordinate systems usually follow the convention that values increase as you move from left to right and from bottom to top. This is not always the case though. Many standard charts that users want to reproduce on the computer have the x-scale, the y-scale, or both, increase as you move from right to left and from top to bottom.

Invert the x- and/or y-scales by swapping the scale starting and ending vaues in the call to the **CartesianCoordinates** or the **TimeCoordinates** constructor.

Example of inverted x-scale using the CartesianCoordinates constructor

[C#]

```
double xMin = -5;  
double xMax = 15;  
double yMin = 0;  
double yMax = 15;  
  
CartesianCoordinates simpleScale;  
simpleScale = new CartesianCoordinates(xMax, yMin, xMin, yMax);
```


[Visual Basic]

```

Dim xMin As Double = -5
Dim xMax As Double = 15
Dim yMin As Double = 0
Dim yMax As Double = 15

Dim simpleScale As CartesianCoordinates
simpleScale = New CartesianCoordinates(xMax, yMin, xMin, yMax)

```

Use the **CartesianCoordinates.SetCoordinateBounds** method in the same manner. The example below inverts the y-scale.

```

simpleScale.SetCoordinateBounds(xMin, yMax, xMax, yMin)

```

Invert the x- and y-scale of a **TimeCoordinates** object in an analogous fashion.

Example of inverted scaling using a TimeCoordinates constructor**[C#]**

```

ChartCalendar xMin = new ChartCalendar(1996, ChartObj.FEBRUARY, 5);
ChartCalendar xMax = new ChartCalendar(2002, ChartObj.JANUARY, 5);
double yMin = 0;
double yMax = 15;

TimeCoordinates simpleTimeScale;
simpleTimeScale = new TimeCoordinates(xMax, yMin, xMin, yMax);

```

[Visual Basic]

```

Dim xMin As ChartCalendar = New ChartCalendar(1996, ChartObj.FEBRUARY, 5)
Dim xMax As ChartCalendar = New ChartCalendar(2002, ChartObj.JANUARY, 5)
Dim yMin As Double = 0
Dim yMax As Double = 15

Dim simpleTimeScale As TimeCoordinates
simpleTimeScale = New TimeCoordinates(xMax, yMin, xMin, yMax)

```

Use the **TimeCoordinates.SetCoordinateBounds** method in the same manner. The example below inverts the y-scale.

[C#]

```
TimeCoordinates simpleTimeScale = new TimeCoordinates();
simpleTimeScale.SetCoordinateBounds(xMin, yMax, xMax, yMin);
```

[Visual Basic]

```
Dim simpleTimeScale As TimeCoordinates = New TimeCoordinates()
simpleTimeScale.SetCoordinateBounds(xMin, yMax, xMax, yMin);
```

The auto-scale functions always create scales that increase from left to right, and bottom to top. This does not exclude the use of the auto-scale functions when creating inverted axes. After an auto-scale function creates the initial x- and y-scales, either or both can be inverted by using the **CartesianCoordinates** or **TimeCoordinates** **InvertScaleX**, **InvertScaleY**, or **InvertScaleZ** methods.

Example of inverting a scale created using the auto-scale methods

[C#]

```
double [] xData = {2,3,4,5,6,7,8,9};
double [] yData = { 22, 33, 44, 55, 46, 33, 25, 14};
double zvalue = 0.0;
double zdepth = 0.1;

SimpleDataset dataset = new SimpleDataset("Sales", xData, yData, zvalue, zdepth);
CartesianCoordinates simpleScale = new CartesianCoordinates();
simpleScale.AutoScale(dataset);
simpleScale.InvertScaleY();
```

[Visual Basic]

```
Dim xData() As Double = {2,3,4,5,6,7,8,9}
Dim yData() As Double = { 22, 33, 44, 55, 46, 33, 25, 14}
Dim zvalue As Double = 0.0
Dim zdepth As Double = 0.1

Dim dataset As SimpleDataset = _
    New SimpleDataset("Sales", xData, yData, zvalue, zdepth)
Dim simpleScale As CartesianCoordinates = New CartesianCoordinates()
simpleScale.AutoScale(dataset)
simpleScale.InvertScaleY()
```

5. The Chart View

ChartView

The starting point of a chart is the **ChartView** class. The **ChartView** class derives from the **System.Windows.Forms.UserControl**, where the **Forms** class is the base class for the .Net collection of standard components such as menus, buttons, check boxes, etc. The **ChartView** class contains a collection of all the chart objects displayed in the chart and will automatically update all chart objects when the underlying window moves, resizes, or otherwise needs to redraw in response to a **UserControl** paint event. Since a **ChartView** derived window is a **UserControl**, it can also be used as a container for any .Net component that can be placed and positioned in **UserControl** windows using a .Net layout managers.

UserControl



The **ChartView** class has only one constructor with no arguments.

ChartView constructor

[Visual Basic]

```
Public Class ChartView
```

```
Inherits UserControl
```

[C#]

```
public class ChartView : UserControl
```

All chart objects that have a graphical representation, i.e. that consist of lines, bars, arcs, text, etc., are subclasses of the **GraphObj** abstract base class. This includes all of the axis classes, axis label classes, plot classes, text classes and legend classes among others. You must explicitly add objects of this type to the **ChartView** object, using the **ChartView.AddChartObject** method, after they have been created and initialized. Otherwise, the object will not be included in the draw list of the **ChartView**. The example below adds an axis object to the **ChartView** draw list.

ChartView.AddChartObject example (extracted from the example program LabeledDatapoints.LabeledDatapoints)

[C#]

144 The ChartView

```
ChartView chartVu = this;
.
.
.

// Create a 3D dataset
SimpleDataset3D Dataset1 = new SimpleDataset3D("First",x1,y1, z1);
// Create a coordinate system to match data
CartesianCoordinates pTransform1 =
    new CartesianCoordinates( ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);

// Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.125, .15, .90, 0.8) ;
.
.
.
// Define x- and y-axes
LinearAxis xAxis = new LinearAxis(pTransform1, ChartObj.X_AXIS);
chartVu.AddChartObject(xAxis);
```

[Visual Basic]

```
Dim chartVu As ChartView = Me
.
.
.
' Create a 3D dataset
Dim Dataset1 As New SimpleDataset3D("First", x1, y1, z1)
' Create a coordinate system to match data
Dim pTransform1 As New _
    CartesianCoordinates(ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

' Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.125, 0.15, 0.9, 0.8)
.
.
.
' Define x- and y-axes
Dim xAxis As New LinearAxis(pTransform1, ChartObj.X_AXIS)
chartVu.AddChartObject(xAxis)
```

Setting the Relative Depth of the ChartView

The relative depth of the z-dimension is set using the **ChartView** method, **SetFractionalZViewportDepth**. It sets the depth of the z-dimension as a fraction of the **ChartView** width. The default value of the z-dimension depth is 0.3, meaning that the apparent depth of the z-dimension will be 30% of the apparent width of the **ChartView**. Making the value larger will give the 3D chart more depth, smaller values mean less depth. The two examples below show how changes in the value can affect the appearance of a chart.

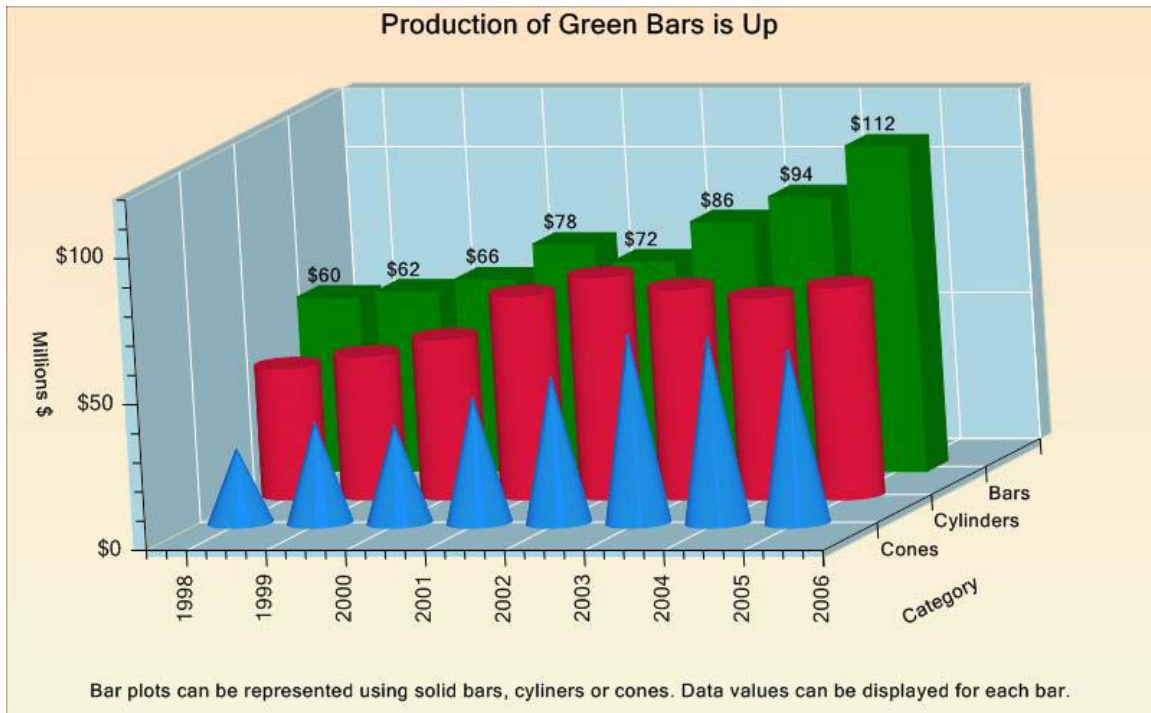
SetFractionalZViewportDepth method

Set the viewport depth as a fraction of the viewport width.

```
[Visual Basic]
Public Sub SetFractionalZViewportDepth( _
    ByVal depth As Double _
)

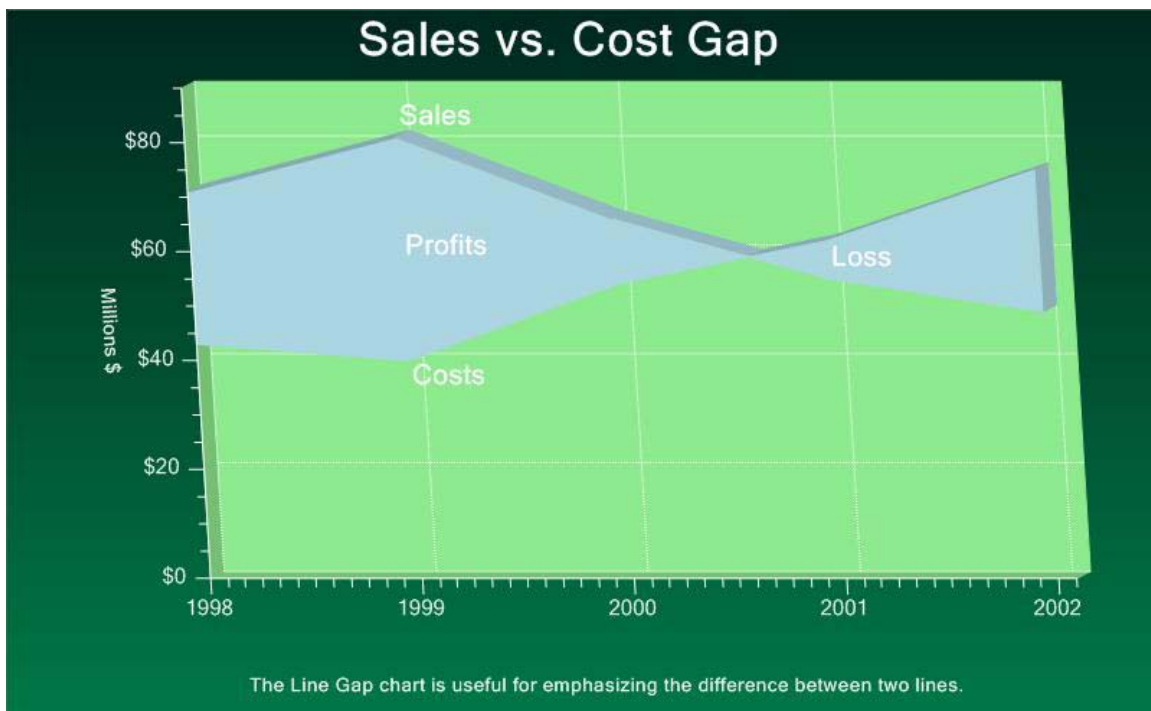
[C#]
public void SetFractionalZViewportDepth(
    double depth
);
```

depth Set the viewport depth as a fraction of the viewport width.



(Bargraphs.SimpleBars)

```
chartVu.SetFractionalViewportDepth(0.8);
```



(*MiscCharts.LineGap*)

```
chartVu.SetFractionalZViewportDepth(0.05);
```

Example of setting the fractional **ChartView** depth to 0.6. Extracted from the example program Bargraphs.SimpleBars.

[C#]

```
// Auto-scale a coordinate system for data
pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(datasetarray, ChartObj.AUTOAXES_FAR ,
    ChartObj.AUTOAXES_FAR);
// force starting y-coordinate scale value of 0
pTransform1.SetScaleStartY(0);
// force starting z-coordinate calendar value
pTransform1.SetTimeScaleStart(new ChartCalendar(1997,ChartObj.JULY,1));

// Set chart rotation

// Make the z-coordinate viewport depth to be 0.6 of the width of the viewport
chartVu.SetFractionalZViewportDepth(0.6);
// Set render quality
chartVu.RenderQuality = ChartObj.RENDER_LOWQUALITY;

// position plot area in graph graph area
pTransform1.SetGraphBorderDiagonal(0.15, 0.2, .8, 0.7) ;
// define a graph background
Background background = new Background( pTransform1, ChartObj.GRAPH_BACKGROUND,
    Color.Beige, Color.Bisque, ChartObj.Y_AXIS);
chartVu.AddChartObject(background);
// set the rotational characteristics of the chart
pTransform1.AbsRotateCoordinateSystem(new Point3D(10,20,0));
```

[VB]

```
' Auto-scale a coordinate system for data
Dim pTransform1 As New TimeCoordinates()
pTransform1.AutoScale(datasetarray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
' force starting y-coordinate scale value of 0
```

148 The ChartView

```
pTransform1.SetScaleStartY(0)
' force starting z-coordinate calendar value
pTransform1.SetTimeScaleStart(New ChartCalendar(1997, ChartObj.JULY, 1))

' Set chart rotation
' Make the z-coordinate viewport depth to be 0.6 of the width of the viewport
chartVu.SetFractionalZViewportDepth(0.6)
' Set render quality
chartVu.RenderQuality = ChartObj.RENDER_LOWQUALITY

' position plot area in graph graph area
pTransform1.SetGraphBorderDiagonal(0.15, 0.2, 0.8, 0.7)
' define a graph background
Dim background As New Background(pTransform1, ChartObj.GRAPH_BACKGROUND,
Color.Beige, Color.Bisque, ChartObj.Y_AXIS)
chartVu.AddChartObject(background)

' set the rotational characteristics of the chart
pTransform1.AbsRotateCoordinateSystem(New Point3D(10, 20, 0))
```


Rendering Order of GraphObj Objects

Each **GraphObj** object is added as an element to an **ArrayList** object inside the **ChartView** class. When the chart view is rendered, it runs through the **GraphObj** objects stored in the list and renders them one by one to the current view. There are two ordering methods used to render chart objects. The first method renders the objects in order, as added to the **ChartView** object. Objects added to the view last are drawn on top of objects added first. The second method renders the objects according to their z-order. Objects with the lowest z-order values are rendered first. Objects with equal z-order values are rendered in the order they are added to the **ChartView** object. The second method (z-order rendering) is the default method of object rendering used by the **ChartView** class. This default behavior can be changed by call the **ChartView.SetZOrderSortEnable(false)** method.

Each **GraphObj** object has a default z-order value, summarized below.

Base Class	Default z-order value	Comments
Background	10	Backgrounds are drawn first. A plot area volume background has a z-value of 10 and a graph area volume background has a z-value of 9, forcing graph area volume backgrounds to be drawn first.
Grid	40	A z-value of 40 places grids under most other graph objects. If you want grids on top change the z-value to 150.
Wall3D	10	A z-value of 10 places walls under most other graph objects, except for Background objects.
GraphObj	50	The default value for graph objects if not explicitly changed in the subclass.
ChartText	50	The default value for text objects.
ChartPlot	50	The default value for plot objects which includes SimplePlot , GroupPlot , and ContourPlot objects.
Axis	100	Chart axes are drawn after data plots
AxisLabels	100	Axes labels are drawn with same priority as axes

Legend	150	Legend objects usually sit on top of all other graph objects and are drawn last

You can change the default z-order value on an object-by-object basis. Call the **GraphObj.SetZOrder** method to change the z-order for any given object.

The example below sets the z-order value of the x-axis to 30, changing the drawing order so that the x-axis draws before, and is therefore underneath, any **Grid** and **ChartPlot** objects in the view.

[C#]

```
ChartView chartVu = new ChartView();

LinearAxis xAxis = new LinearAxis(pTransform1, ChartObj.X_AXIS);
xAxis.SetZOrder(30);
chartVu.AddChartObject(xAxis);
```

[Visual Basic]

```
Dim chartVu As ChartView = New ChartView()

Dim xAxis As LinearAxis = New LinearAxis(pTransform1, ChartObj.X_AXIS)
xAxis.SetZOrder(30)
chartVu.AddChartObject(xAxis)
```

Dynamic or Real-Time Updates of Chart Objects

If you want to change the properties of one or more **GraphObj** derived objects displayed in the current graph, just go ahead and change them using the appropriate Get and Set methods. Once you change all of the properties that you want, call the **ChartView.UpdateDraw** method. This will force the **ChartView** object to update, redrawing every object in its draw list. See the example below.

[C#]

```
ChartView chartVu = new ChartView();

LinearAxis xAxis = new LinearAxis(pTransform1, ChartObj.X_AXIS);
chartVu.AddChartObject(xAxis);

.
.
.
```

```
// Change the color of the x-axis after it has been created
xAxis.SetColor(Color.Red);
chartVu.UpdateDraw();
```

[Visual Basic]

```
Dim chartVu As ChartView = New ChartView()

Dim xAxis As LinearAxis = New LinearAxis(pTransform1, ChartObj.X_AXIS)
chartVu.AddChartObject(xAxis)

.
.
.
xAxis.SetColor(Color.Red)
chartVu.UpdateDraw()
```

You can change the values of a dataset, or even change the complete dataset of a chart plot object. Changing the values of the dataset will not show in the current graph until the **ChartView** repaints. Call the **ChartView.UpdateDraw** method to force a repaint. See the example programs DynPieChart and ScrollingMixedPlot. The auto-scale methods are not automatically invoked if you change the values of dataset. If you want the graph to rescale taking into account the new data values you must call the appropriate autoscale methods of the coordinate system and of the related axes objects.

The chart classes that are NOT subclasses of **GraphObj** do not have a physical representation in a graph, so do not try to add them to the **ChartView** draw list. This includes the coordinate conversion classes, the dataset classes and all of the utility classes. The **GraphObj** and **ChartView** classes use these utility classes for coordinate conversions, data storage, math calculations and I/O.

Delete a specific chart object from the **ChartView** draw list using the **ChartView.DeleteChartObject** method. Clear the entire draw list using the **ChartView.ResetChartObjectList** method. You can leave an object in the **ChartView** draw list but disable its display by calling that objects **GraphObj.SetChartObjEnable** method. If you disable an object, you will still need to call the **ChartView.UpdateDraw** method to redraw the chart without that object.

Placing Multiple Charts in a ChartView

One way to create multiple charts is to create multiple instances of the **ChartView** class and add each **ChartView** object to a .Net container object such as a **UserControl**. A .Net

layout manager manages the position and size of each **ChartView**. The **WebsiteStats** example program demonstrates simple placement of four **ChartView** objects on a form. Another way is to place multiple charts in the same **ChartView** object. This makes it easier to guarantee alignment between the axes of separate graphs. The trick to doing this is to create separate coordinate system objects (**CartesianCoordinates**, or **TimeCoordinates**) for each chart, and to position the plot area volume of each coordinate system so that they do not overlap. Use one of the coordinate systems **SetGraphBorder...** methods. Many of the examples use this technique, including **GroupBarPlotChart**, **DoubleBarPlot**, **OHLFinPlot**, **FinOptions**, **DynPieChart**, **PieAndLineChart** and **PieAndBarChart**.

Multiple charts in a **ChartView** example (extracted from the example program **FinancialExamples.OHLCChart**)

[C#]

```
pTransform1 = new TimeCoordinates();
pTransform1.SetGraphBorderDiagonal(0.1, .15, .90, 0.55) ;

pTransform2 = new TimeCoordinates();
pTransform2.SetGraphBorderDiagonal(0.1, .675, .90, 0.85) ;
```

[Visual Basic]

```
pTransform1 = New TimeCoordinates()
pTransform1.SetGraphBorderDiagonal(0.1, .15, .90, 0.6)
pTransform2 = New TimeCoordinates()
pTransform2.SetGraphBorderDiagonal(0.1, .7, .90, 0.875)
```

Multiple Coordinate Systems in the Same Chart

Often a chart needs more than one coordinate system to in order to support multiple x- and y-axes, each with different scales. As in the preceding section, this involves creating multiple coordinate systems. The plot area volumes for the coordinate systems can occupy separate space in the chart view, or they can overlap at the exact same position. As in the previous section, the position of each coordinate systems plot area volume in the chart view is set using one of the coordinate systems **SetGraphBorder...** methods. Many of the examples use this technique, including **FinancialExamples.OHLCChart**, **MultipleAxes.MultipleAxes**, **ChartAxes.LinearAxes**, **ChartAxes.LogAxes** and **ChartAxes.DateAxes1** and **ChartAxes.DateAxes2**.

Multiple coordinate systems in a ChartView example**[C#]**

```

double xMin1 = -5;
double xMax1 = 15;
double yMin1 = 0;
double yMax1 = 105;
CartesianCoordinates pTransform1 =
    new CartesianCoordinates(xMin1, yMin1, xMax1, yMax1);
pTransform1.SetGraphBorderDiagonal(0.1, .15, .90, 0.6) ;

double xMin2 = -50;
double xMax2 = 150;
double yMin2 = 0;
double yMax2 = 1050;
CartesianCoordinates pTransform2 =
    new CartesianCoordinates(xMin2, yMin2, xMax2, yMax2);
pTransform2.SetGraphBorderDiagonal(0.1, .15, .90, 0.6) ;

```

[Visual Basic]

```

Dim xMin1 As Double = -5
Dim xMax1 As Double = 15
Dim yMin1 As Double = 0
Dim yMax1 As Double = 105
Dim pTransform1 As CartesianCoordinates = _
    New CartesianCoordinates(xMin1, yMin1, xMax1, yMax1)
pTransform1.SetGraphBorderDiagonal(0.1, .15, .90, 0.6)

Dim xMin2 As Double = -50
Dim xMax2 As Double = 150
Dim yMin2 As Double = 0
Dim yMax2 As Double = 105
Dim pTransform2 As CartesianCoordinates = _
    New CartesianCoordinates(xMin2, yMin2, xMax2, yMax2)
pTransform2.SetGraphBorderDiagonal(0.1, .15, .90, 0.6)

```

ChartView Object Resize Modes

Every **GraphObj** object has absolute size properties, such as font size or line thickness. Resize a window and these absolute size parameters are NOT changed. No matter how

you resize a chart, if you set a text object to a font size of 10, the text object will always return a font size of 10, *regardless if the text now appears larger or smaller*. Instead, the value of the **resizeMultiplier** adjusts to represent the proportional change in the window size. In calculating the font size and the line thickness, the current size properties are multiplied by the **resizeMultiplier**. The initial value of the **resizeMultiplier** is 1.0 and the objects size properties correspond exactly to the initial settings. Shrink the **ChartView** window and the **resizeMultiplier** for each object is set to a value that is less than 1.0. Enlarge the window and the **resizeMultiplier** is set to a value greater than 1.0. The **ChartView** class has three resize modes that it can use to resize graph objects placed in the graph.

NO_RESIZE_OBJECTS The **resizeMultiplier** stays fixed at 1.0. Resizing the graph window does not affect the size and thickness of the charts graph objects. Text will stay the same size and lines will stay the same thickness. The overall chart shrinks; it is just that size of the chart text and the thickness of the chart lines do not change. Resize the window small enough and the chart text will overlap and the lines used to draw the chart will look thick when compared to the chart size.

AUTO_RESIZE_OBJECTS Resizing the graph window causes the size and thickness of the charts graph objects to resize. The auto-resize algorithm looks at which dimension changed the most (x or y) and uses the larger of the two changes to calculate new sizes for lines and text. Text and lines will shrink the same percentage. Resize the chart window with a minimum change in the charts aspect ratio, the change in the chart size will be very close to the change in the font size and line thickness. If the charts aspect ratio changes drastically, the font size and line thickness will resize to reflect the dimension that was *reduced* the most. This minimizes the condition where text overlaps, though it may make the text unreadable if the chart size changes from large to a small with a large aspect ratio change.

MANUAL_RESIZE_OBJECTS

The **resizeMultiplier** for each object has an initial value of 1.0. Unlike the **NO_RESIZE_OBJECTS** mode, the value can be changed. The programmer must explicitly set the **resizeMultiplier** for any objects requiring a size change, using the **GraphObj.SetResizeMultiplier** method.

ChartView View Modes

A **ChartView** window can interact with a parent container, creating a couple of interesting view modes. The first is to place the **ChartView** object in a Windows Form. Size the **ChartView** window larger than the Form window size and the Form frame acts as a “porthole” through which the **ChartView** window is viewed. Resizing the Form frame has no effect on the size of **ChartView** object within. If the Form frame is smaller than the size of the **ChartView** window then the lower right portion of the **ChartView** window is not visible. If the Form frame is larger than the **ChartView** window, the lower right portion of the window just shows empty space.

Another interesting technique is to place a fixed size **ChartView** object in a Windows Form that has the `AutoScroll` property set to true. The size of the **ChartView** window can be much larger than the Form window size. Place the chart in a form that has the `AutoScroll` property set to true. The auto-scroll feature of the Form displays scroll bars when the chart object is larger than the containing form. The scroll bars permit the user to pan left, right, up and down to view the portion of the **ChartView** window that is outside of the clipping limits of the Form window. See the `FixedSizeScrollable` tab of the `ResizeExamples` program for an example.

There are many more variations. Just remember that the **ChartView** class is a **UserControl** derived class and it can be used anywhere a **UserControl** object can be used, utilizing standard or specialized .Net layout managers.

Finding Chart Objects

The **ChartView** class is the central container class of the chart library. It keeps track of all of the objects in the chart. It includes a routine that can compare a test point against all of the objects in the chart and return an instance of an object that intersects the test point. The search can be restricted to a class and all subclasses of the specified class.

FindObj method

```
[Visual Basic]
Overloads Public Function FindObj( _
    ByVal testpoint As Point3D, _
    ByVal classname As String _
) As GraphObj
```

[C#]

```
public GraphObj FindObj(
    Point3D testpoint,
    string classname
```

```
);
```

Parameters

If the graph has multiple overlapping objects of the same type, you can return the n^{th} object intersecting the test point.

```
[Visual Basic]
Overloads Public Function FindObj( _
    ByVal testpoint As Point3D, _
    ByVal classname As String, _
    ByVal nthhit As Integer _
) As GraphObj
```

```
[C#]
```

```
public GraphObj FindObj(
    Point3D testpoint,
    string classname,
    int nthhit
);
```

<i>testpoint</i>	The current position of the mouse in .Net device coordinates.
<i>classname</i>	The class name of the base class that is used to filter the desired class objects. The string "ChartPlot" would cause the routine to return only objects derived from the ChartPlot class.
<i>nthhit</i>	Specifies to return the n^{th} object that intersects the test point. A value of 0 signifies that the first object found is returned, a value of 1 specifies that the second item found is returned, and so on.

The function returns a reference to the found object, or null if unsuccessful.

6. Backgrounds, Walls, Borders and Bitmaps

There are four ways to add background elements to your charts. The first is to use the **Background** class to add a color, or color gradient to the entire **ChartView** viewport of the chart. The second is to use the **Background** class to provide background planes for the plot area volume of a specific coordinate system. The third is to use one or more **Wall3D** objects to provide 3-dimensional walls as the background for the plot area volume of a coordinate system. The last is to place a bitmap as a background for a chart.

Background

There are two types of background elements that can serve as a backdrop for the other graphical objects of a chart. The first is simple 2D rectangle formed by the **ChartView** viewport, also called the *graph area*. This rectangle is the entire area of the **ChartView** window, and all elements of the chart (axes, labels, plots, titles, walls etc.) are within its bounds. The second area is the *plot area* volume. The plot area volume is the 3D prism within the **ChartView** viewport that defines the limits of the physical coordinate system of the chart. Its position within the graph area volume is set using one of the **PhysicalCoordinates.SetGraphBorder...** methods: **SetGraphBorderDiagonal**, **SetGraphBorderFrame** or **SetGraphBorderInsets**. Typically the chart plot objects (line plots, bar plots, scatter plots, etc.) are clipped to the plot area volume. Other chart objects (axes, axes labels, titles, etc.) need to reside outside of the plot area volume and these objects clip to the graph area volume. The plot area volume can have a background different from that of the graph background. Often a contrast between the graph area volume background and the plot area volume background produces a more visually pleasing chart.

Class Background

GraphObj

|
+-- Background

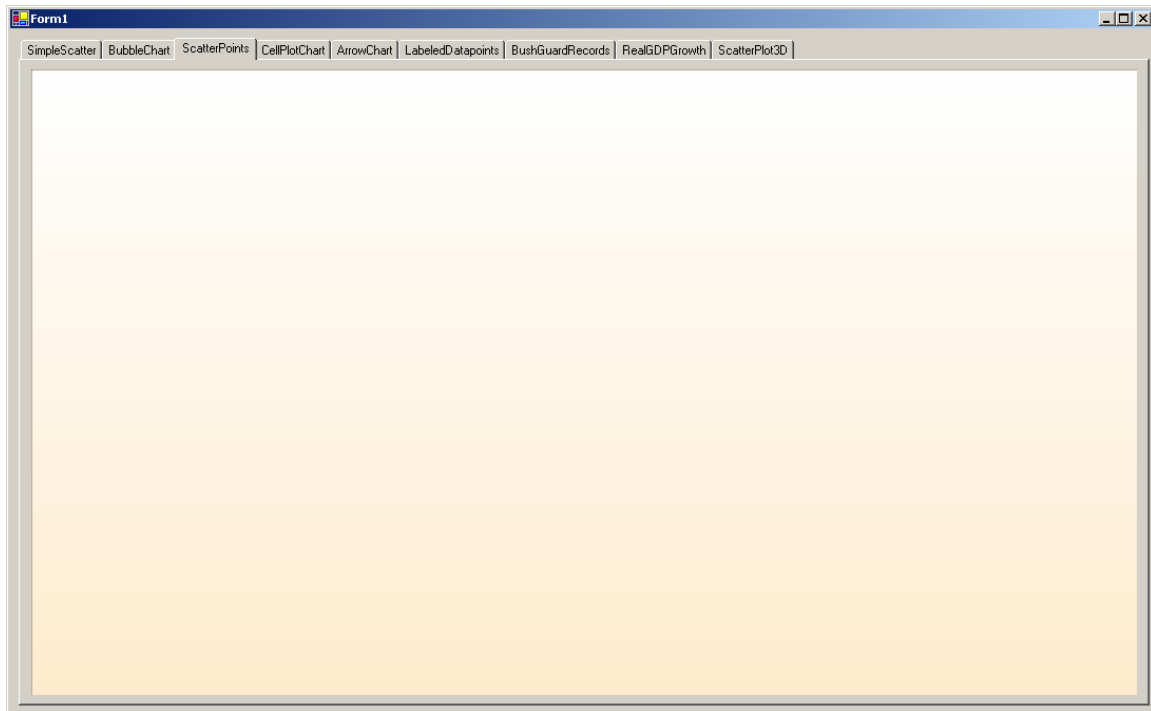
The **Background** class paints the graph area background or the plot area volume background. One instance of the class can only paint one area, either the graph area volume or the plot area volume. If you want unique fill properties for both, you need to create two instances of the class. In the graph area background mode, the **Background** class uses one of the following techniques to fill the background:

- solid color
- simple color gradient defined using two RGB colors

- user-defined gradient supplied as a .Net **LinearGradientBrush** object
- user-defined texture supplied as a .Net **TextureBrush** object

The background color and any gradients you apply is independent of any rotations that you apply to the coordinate system and will not be shaded.

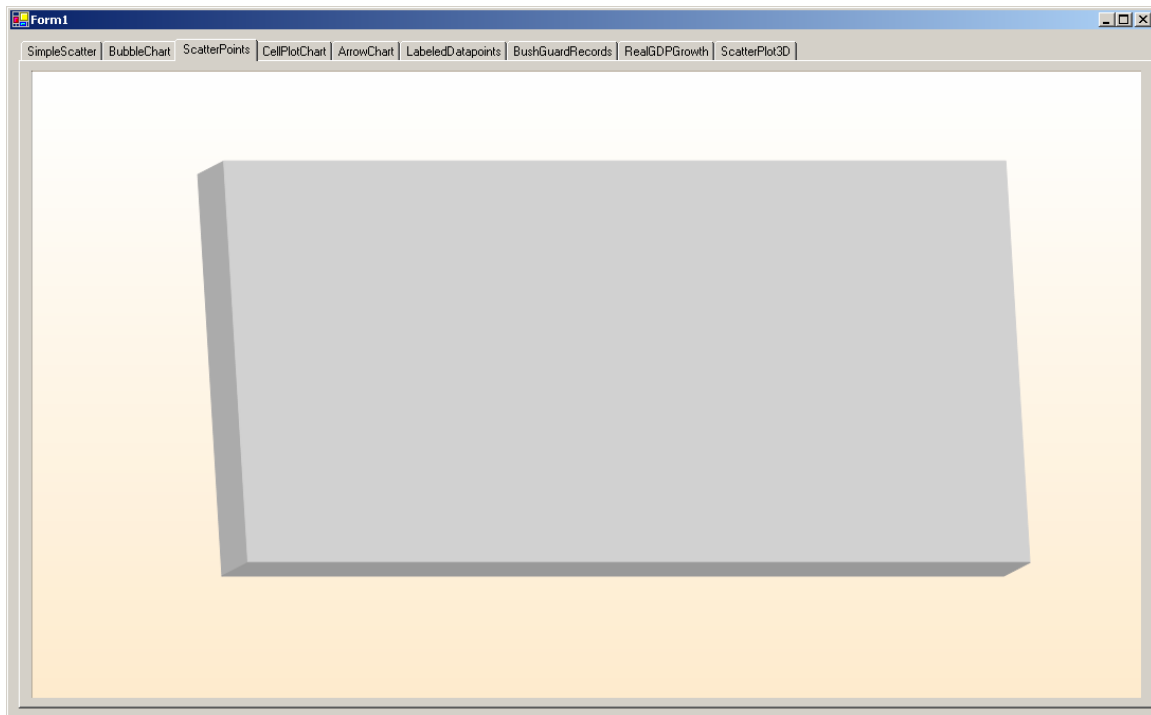
```
Background background = new Background( pTransform1,
    ChartObj.GRAPH_BACKGROUND, Color.BlanchedAlmond, Color.White, ChartObj.Y_AXIS);
```



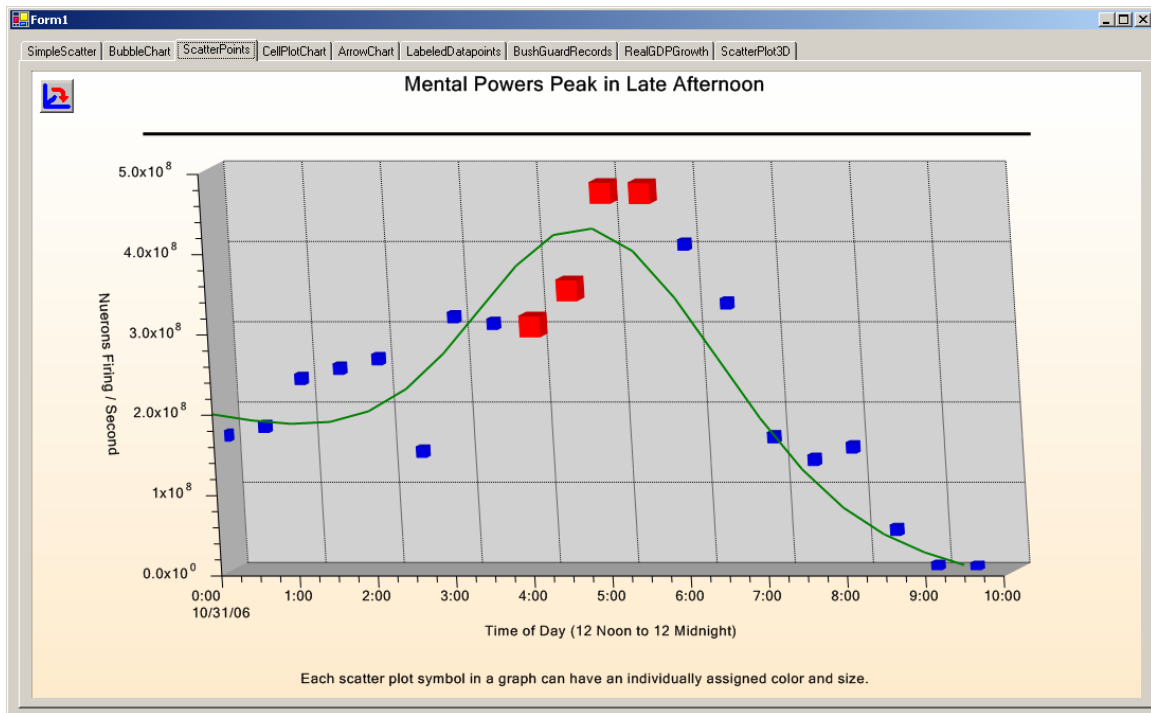
A chart area background with y-axis gradient:

The plot area background mode always uses a simple solid color. By default, three planes of the plot area 3D volume are displayed. These planes are the back plane (XY_MAXZ_PLANE), the left plane (YZ_MINX_PLANE) and the bottom plane (XY_MINY_PLANE). You can turn on/off the individual planes using the PlotAreaWallFlags property. The plot area background planes are dependent on the current coordinate system 3D rotations and will be shaded accordingly.

```
Background plotbackground = new Background( pTransform1,
    ChartObj.PLOT_BACKGROUND, Color.LightGray);
plotbackground.PlotAreaWallFlags[ChartObj.XY_MAXZ_PLANE] = true;
plotbackground.PlotAreaWallFlags[ChartObj.YZ_MINX_PLANE] = true;
plotbackground.PlotAreaWallFlags[ChartObj.XZ_MINY_PLANE] = true;
```



A plot area background, LightGray in color, shaded according to the chart rotation



*This example uses both a graph area background and a plot area background.
(ScatterPlots.ScatterPoints)*

Background constructors

Use this constructor to fill the background with a single color.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal bgtype As Integer, _
    ByVal bgcolor As Color _
)

[C#]
public Background(
    PhysicalCoordinates transform,
    int bgtype,
    Color bgcolor
);
```

Use this constructor to fill the background with the gradient defined using the *startcolor* and *stopcolor* arguments.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal bgtype As Integer, _
    ByVal startcolor As Color, _
    ByVal stopcolor As Color, _
    ByVal dir As Integer _
)

[C#]
public Background(
    PhysicalCoordinates transform,
    int bgtype,
    Color startcolor,
    Color stopcolor,
    int dir
);
```

Use this constructor to fill the background with a user-defined gradient.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal bgtype As Integer, _
    ByVal gradient As LinearGradientBrush _
)

[C#]
public Background(
    PhysicalCoordinates transform,
    int bgtype,
    LinearGradientBrush gradient
);
```

Use this constructor to fill the background with a user-defined texture.

```
[Visual Basic]
Overloads Public Sub New( _
```

```

    ByVal transform As PhysicalCoordinates, _
    ByVal bgtype As Integer, _
    ByVal texture As TextureBrush _
)
[C#]
public Background(
    PhysicalCoordinates transform,
    int bgtype,
    TextureBrush texture
);

```

<i>transform</i>	The coordinate system associated with the chart background. The <i>transform</i> defines where the plot area volume fits in the graph area volume.
<i>bgtype</i>	The chart background type. Use one of the chart background type constants: PLOT_BACKGROUND or GRAPH_BACKGROUND. Specifying the PLOT_BACKGROUND type fills the plot area volume of the chart, while specifying the GRAPH_BACKGROUND type fills the entire graph area volume of the chart.
<i>gradient</i>	The user defined background gradient.
<i>texture</i>	The users defined background texture.
<i>startcolor</i>	Specifies the starting color value of the gradient.
<i>stopcolor</i>	Specifies the ending color value of the gradient.
<i>dir</i>	Specifies the direction of the gradient.

Bitmaps as Background Objects

If you want to use some sort of image as a background for the chart, use the **ChartImage** class and size it to fill the entire view. This is explained in Chapter 21 – Adding Lines, Shapes, Images and Arrows to a Chart.

Add a Simple Borders to Your Chart

The **Background** class also draw an optional border rectangle, surrounding the **ChartView** at the very outside edge of its bounding region. Enable the border using the **Background.EnableBorder** property. The property **Background.BorderAttributes** sets set the line color and thickness of the border.

```

[VB]
' Define background
Dim background As New Background(pTransform1, _

```

```

        ChartObj.GRAPH_BACKGROUND, Color.White)
background.EnableBorder = True
background.BorderAttributes = New ChartAttribute(Color.Gray, 5, DashStyle.Solid)
chartVu.AddChartObject(background)

[C#]
// Define background
Background background =
    new Background( pTransform1, ChartObj.GRAPH_BACKGROUND, Color.White);
background.EnableBorder = true;
background.BorderAttributes = new ChartAttribute(Color.Gray, 5, DashStyle.Solid);
chartVu.AddChartObject(background);

```

Every example program configures at least one Background object. The example below defines a simple linear gradient for the graph background area (extracted from the example program SimpleLinePlots.LineFill)

```

[C#]

// Create a coordinate system scaled for the data
pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(DatasetArray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);

// Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, .1, .92, 0.85) ;
.
.
.
// Background for graph
Background background = new Background( pTransform1,
ChartObj.GRAPH_BACKGROUND,
    Color.White, Color.LightGray, ChartObj.Y_AXIS);
chartVu.AddChartObject(background);

```

[Visual Basic]

```

' Create a coordinate system scaled for the data
pTransform1 = New TimeCoordinates()
pTransform1.AutoScale(DatasetArray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

' Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, 0.1, 0.92, 0.75)

```

```

.
.
.

' Background for graph
Dim background As New Background(pTransform1, ChartObj.GRAPH_BACKGROUND, _
    Color.White, Color.LightGray, ChartObj.Y_AXIS)
chartVu.AddChartObject(background)

```

Class Wall3D

GraphObj

```

|
+-- Wall3D

```

The **Wall3D** class draws a single 3D wall in graph area using the specified coordinate system and drawing attributes. The wall can be positioned at one of the planes associated with the coordinate system (XY_MINZ_PLANE, XZ_MINY_PLANE, YZ_MINX_PLANE, XY_MAXZ_PLANE, XZ_MAXY_PLANE, YZ_MAXX_PLANE). Only one wall is drawn for each **Wall3D** object. If you want three walls at three separate planes, as most of our examples do, you need to create three unique **Wall3D** objects. A **Wall3D** object is different from a **Background** object in the PLOT_BACKGROUND mode, because the **Background** planes are 2-dimensional and have no depth to them. A **Wall3D** object is drawn as a 3D prism, with height, width and depth.

Wall3D constructors

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal wallorientation As Integer, _
    ByVal wallthickness As Double, _
    ByVal wallattrib As ChartAttribute _
)

[C#]
public Wall3D(
    PhysicalCoordinates transform,
    int wallorientation,
    double wallthickness,
    ChartAttribute wallattrib
);

```

transform The scale associated with the chart background.

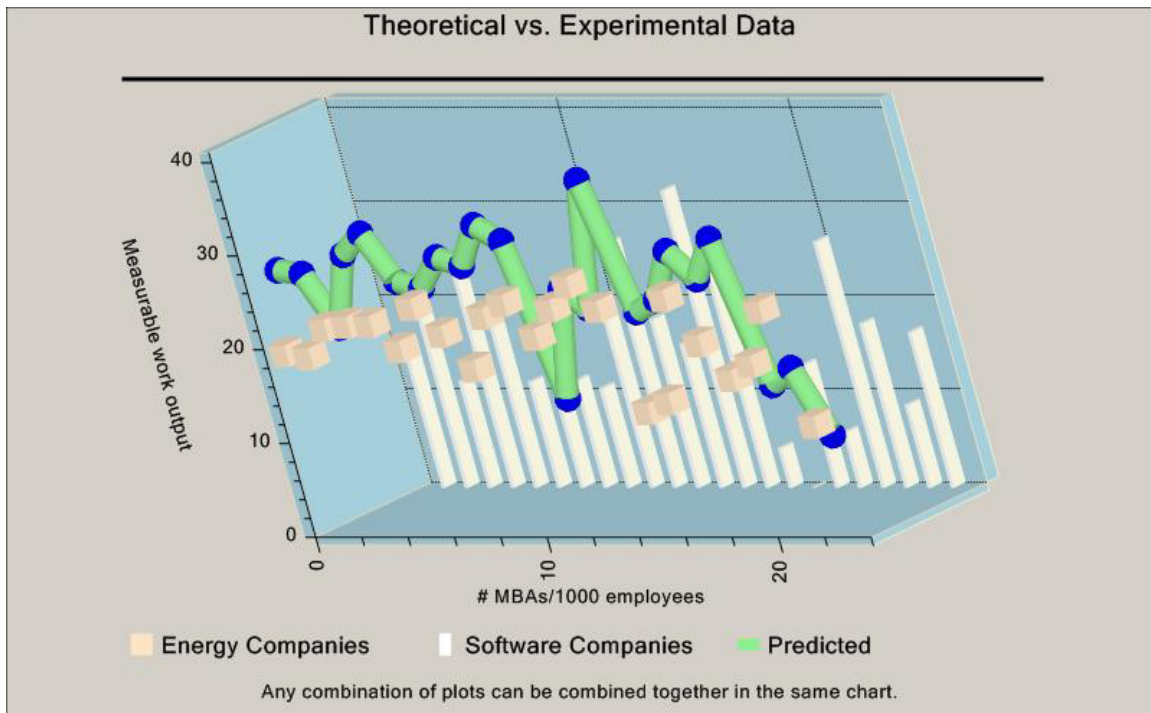
wallorientation

Specifies the position of the wall. Use one of the ChartSupport wall orientation constants: XY_MINZ_PLANE, XZ_MINY_PLANE, YZ_MINX_PLANE, XY_MAXZ_PLANE, XZ_MAXY_PLANE, YZ_MAXX_PLANE.

wallthickness Specifies the thickness of the wall in normalized (0.0 to 1.0) coordinates.

wallattrib The **ChartAttribute** of the wall.

Simple Wall3D example (extracted from the example program SimpleLinePlots.PipeLineAndScatterPlot)



(SimpleLinePlots.PipeLineAndScatterPlot)

[VB]

```
' Define coordinate system based on data
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE,
ChartObj.LINEAR_SCALE)
```



```

pTransform1.AutoScale(Dataset3, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
' Add a bit to the auto-calculated y-scale
pTransform1.ScaleStopY += 1.0

' Set rotational parameters
pTransform1.AbsRotateCoordinateSystem(New Point3D(10, 10, 0))
' Set chart depth
chartVu.SetFractionalZViewportDepth(0.2)

' Place chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 0.725)

' Define chart background
Dim background As New _
    Background(pTransform1, ChartObj.PLOT_BACKGROUND, Color.White)
chartVu.AddChartObject(background)

' Define chart walls
Dim wallAttrib As New ChartAttribute(Color.Beige, 1, _
    DashStyle.Solid, Color.LightBlue)
Dim xyMinZWall As New Wall3D(pTransform1, ChartObj.XY_MAXZ_PLANE, _
    0.1, wallAttrib)
chartVu.AddChartObject(xyMinZWall)
Dim yzMinXWall As New Wall3D(pTransform1, ChartObj.YZ_MINX_PLANE, _
    0.02, wallAttrib)
chartVu.AddChartObject(yzMinXWall)
Dim xzMinYWall As New Wall3D(pTransform1, ChartObj.XZ_MINY_PLANE, _
    0.02, wallAttrib)
chartVu.AddChartObject(xzMinYWall)

```

[C#]

```

// Define coordinate system based on data
CartesianCoordinates pTransform1 =
    new CartesianCoordinates( ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset3, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
// Add a bit to the auto-calculated y-scale
pTransform1.ScaleStopY += 1.0;

// Set rotational parameters
pTransform1.AbsRotateCoordinateSystem(new Point3D(10,10,0));
// Set chart depth
chartVu.SetFractionalZViewportDepth(0.2);

```

```

// Place chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, .15, .90, 0.725) ;

// Define chart background
Background background =
    new Background( pTransform1, ChartObj.PLOT_BACKGROUND, Color.White);
chartVu.AddChartObject(background);

// Define chart walls
ChartAttribute wallAttrib =
    new ChartAttribute (Color.Beige, 1,DashStyle.Solid, Color.LightBlue);
Wall3D xyMinZWall =
    new Wall3D(pTransform1, ChartObj.XY_MAXZ_PLANE, 0.1, wallAttrib);
chartVu.AddChartObject(xyMinZWall);
Wall3D yzMinXWall =
    new Wall3D(pTransform1, ChartObj.YZ_MINX_PLANE, 0.02, wallAttrib);
chartVu.AddChartObject(yzMinXWall);
Wall3D xzMinYWall =
    new Wall3D(pTransform1, ChartObj.XZ_MINY_PLANE, 0.02, wallAttrib);
chartVu.AddChartObject(xzMinYWall);

```

7. Axes

Axis

LinearAxis

LogAxis

TimeAxis

Chart axes describe for the viewer the physical coordinate system used to scale the plot area volume of a chart. A well-defined, visually appealing chart will display one or more axes with the following characteristics:

- Minimum and maximum values for axes endpoints that are appropriate for the displayed data
- Appropriately spaced axis tick marks that permit the user to easily interpolate by simple inspection data values located between labeled tick marks
- Axis tick mark labels that fall on logical, even intervals
- Flexible axis placement, inside or outside the plot area volume
- Axes for linear, time, and logarithmic physical coordinate systems

The programmer can explicitly set these characteristics, or they can be calculated automatically based on an analysis of the associated chart data.

The axes of a chart do not define the physical coordinate system of the chart. Rather, the axes provide a visual key to the physical coordinate system. Define the physical coordinate system first using one of the classes derived from **PhysicalCoordinates**. Next, create the axes that reside in the physical coordinate. It is possible to define a physical coordinate system scaled using a xy range of (0-100, 0-100) and create an axis, residing in that coordinate system, that has minimum and maximum values of (0-25). The axis in that case takes up 25% of the chart plot area volume of the chart. Define the same axis with minimum and maximum values of (0-100) and the axes will span 100% of chart plot area volume.

A chart axis consists of at least two and usually three parts: the axis line, the axis tick marks, and the axis labels. The axis line extends from the minimum value to the maximum value of the axis. Major tick marks perpendicular to the axis line divide the axis line into sub ranges suitable for labeling. Minor tick marks, also perpendicular to the axis line, further subdivide the space between the major tick marks into even smaller intervals. Axis labels are optional. On one side of a chart there may be an axis with labels and on the other side an axis without labels.

Chart Axes

There are three concrete axis types supported by the **QCChart3D for .Net** library:

Axis Type	Class
Linear	LinearAxis
Logarithmic	LogAxis
Date/time	TimeAxis

The three axis types derive directly or indirectly from the **Axis** abstract base class that provides a core set of properties and methods.

All axis objects use the same set of methods, found in the base **GraphObj** class, to set the drawing properties of the lines used to draw the axis line and tick marks. The default values use a black solid line of thickness 1.0. Change the default values using the **GraphObj** methods below.

SetColor method

```
[Visual Basic]
Overridable Public Sub SetColor( _
    ByVal rgbcolor As Color _
)

[C#]
public virtual void SetColor(
    Color rgbcolor
);
```

SetLineWidth method

```
[Visual Basic]
Overridable Public Sub SetLineWidth( _
    ByVal linewidth As Double _
)

[C#]
public virtual void SetLineWidth(
    double linewidth
);
```

SetLineStyle method

```
[Visual Basic]
Overridable Public Sub SetLineStyle( _
    ByVal linestyle As DashStyle _
)

[C#]
public virtual void SetLineStyle(
```

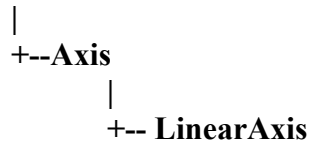
```
        DashStyle linestyle
    );
```

<i>rgbcolor</i>	Sets the primary line color for the chart object.
<i>linewidth</i>	Sets the line width, in device coordinates, for the chart object.
<i>linestyle</i>	Sets the line style for the chart object. Use one of the .Net Drawing2D.DashStyle enumerated constants: Dash, DashDot, DashDotDot, Dot or Solid.

Linear Axes

Class LinearAxis

GraphObj



Linear Axis Minimum and Maximum

The axes minimum and maximum are the physical coordinate values that define the starting and ending points of the axis line. It is a mistake to try to invert the axis, i.e. an axis where the scale decreases from left to right, or bottom to top, by setting axis minimum to a value greater than the axis maximum. The software swaps the values if this happens. Create an inverted axis by first defining an inverted physical coordinate system using one of the **PhysicalCoordinates** derived classes. Place the axis in the inverted coordinate system.

The minimum and maximum of a linear axis can assume any numeric values. This differentiates the linear axis from logarithmic, and time axes that have specific numeric ranges for which they are valid.

Linear Minor and Major Tick Mark Intervals

Major tick marks perpendicular to the axis line divide the line into sub ranges suitable for labeling. Minor tick marks, also perpendicular to the axis line, further subdivide the space between the major tick marks into even smaller intervals.

The major tick mark interval for a linear axis is set equal to a specified integer number of minor tick intervals, forcing major tick marks to always fall on a minor tick mark.

It is important that the tick mark intervals fall on rounded values appropriate to the physical scale of the chart. It is not appropriate to look at the range (maximum value –

minimum value) and divide by some integer. For example, an axis with endpoints -5 to 30 should have a major tick mark interval of 5 or 10 , and a minor tick mark interval 1.0 . Dividing the axis range ($30 - (-5) = 35.0$) by 10 will result in a tick interval of 3.5 , which is inappropriate for either major or minor tick intervals. The programmer can calculate the proper tick mark intervals using custom algorithms, or use the automatic methods that are used by default in the axis constructors.

Linear Axis Intercepts

An axis resides in a 3-dimensional physical coordinate system. The minimum and maximum values for the axis provide coordinate information for only one dimension; x-coordinates in the case of an x-axis, and y-coordinates in the case of the y-axis. The missing coordinates needed to position the axis are the axis intercepts. The axis intercepts specifies the y- and z-coordinate position for the x-axis, the x- and z-coordinate position for the y-axis and the x- and y-coordinate position for the z-axis.

Linear Axis Tick Mark Origin

The axis major and minor tick mark intervals specify the space between adjacent tick marks. A minor tick mark interval of 1.0 , and a major tick mark interval of 5.0 , may result in major tick marks at 0.0 , 5.0 , 10.0 , 15.0 , 20.0 , etc. It can also result in major tick marks at -0.88769 , 4.11231 , 9.11231 , 14.11231 , 19.11231 , etc. Obviously, the first example is the desired tick mark placement. The difference between the two examples is the tick mark starting point, or origin. In the first example, the tick mark origin is 0.0 and in the second case the tick mark origin is -0.88769 . The tick mark origin is an important property because often it should not be the minimum value of the axis, but rather some intermediate value between the minimum and maximum value of the axis. In the example above, the data may range from -0.88769 to 19.9 and the chart is to have exactly that range. It is still appropriate that the tick mark origin be set to 0.0 , rather than the axis minimum value of -0.88769 .

The tick mark origin should reside in the bounds defined by the axis minimum and maximum, inclusive of the endpoints. It does not need to be near an endpoint however. For example, an axis with endpoints -16 to $+19$ should use a minor tick mark interval of 1.0 or 2.0 , a major tick mark interval of 5.0 or 10.0 , and a tick mark origin of 0.0 . Usually, if the axis minimum and maximum bracket 0.0 , i.e. the axis minimum is less than or equal to 0.0 and the axis maximum is greater than or equal to 0.0 , the best tick mark origin to use is 0.0 .

Creating a Linear Axis

There are two main constructors for **LinearAxis** objects. The first **LinearAxis** constructor assumes that the axis extents match the extents of the underlying coordinate system, *transform*. The second **LinearAxis** constructor sets the axis extents to the specified minimum and maximum values, regardless of the underlying coordinate system.

LinearAxis constructors

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal axtype As Integer _
)
```

```
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal axtype As Integer, _
    ByVal minval As Double, _
    ByVal maxval As Double _
)
```

```
[C#]
public LinearAxis(
    PhysicalCoordinates transform,
    int axtype
);
```

```
public LinearAxis(
    PhysicalCoordinates transform,
    int axtype,
    double minval,
    double maxval
);
```

- transform* Places the axes in the coordinate system defined by transform.
- axtype* Specifies if the axis is an x-axis (X_AXIS), y-axis (Y_AXIS) or a z-axis (Z_AXIS).
- minval* Sets the minimum value for the axis.
- maxval* Sets the maximum value for the axis.

Other axis properties: minor tick mark spacing, number of minor tick marks per major tick mark, axis intercept, tick mark lengths, tick mark direction and axis tick mark origin are automatically calculated using an auto-axis method. Set these properties explicitly if you need to override the automatically calculated values.

SetAxisIntercept and SetAxisIntercept2 methods

The **SetAxisIntercept** and **SetAxisIntercept2** methods set the two intercepts needed to position an axis. The coordinates they apply to depends on the axis.

- X_AXIS **SetAxisIntercept** sets the y-intercept of the x-axis. **SetAxisIntercept2** sets the z-intercept of the x-axis.

Y_AXIS **SetAxisIntercept** sets the x-intercept of the y-axis. **SetAxisIntercept2** sets the z-intercept of the y-axis.

Z_AXIS **SetAxisIntercept** sets the x-intercept of the z-axis. **SetAxisIntercept2** sets the y-intercept of the z-axis.

```
[Visual Basic]
Public Sub SetAxisIntercept( _
    ByVal intercept As Double _
)
Public Sub SetAxisIntercept2( _
    ByVal intercept As Double _
)
```

```
[C#]
public void SetAxisIntercept(
    double intercept
);
public void SetAxisIntercept2(
    double intercept
);
```

SetAxisTicks method

```
[Visual Basic]
Overloads Public Sub SetAxisTicks( _
    ByVal tickorigin As Double, _
    ByVal tickspace As Double, _
    ByVal ntickspermajor As Integer _
)
```

```
[Visual Basic]
Overloads Public Sub SetAxisTicks( _
    ByVal tickorigin As Double, _
    ByVal tickspace As Double, _
    ByVal nminortickspermajor As Integer, _
    ByVal minorticklength As Double, _
    ByVal majorticklength As Double, _
    ByVal tickdir As Integer _
)
```

```
[C#]
public void SetAxisTicks(
    double tickorigin,
    double tickspace,
    int ntickspermajor
);
```

```
public void SetAxisTicks(
    double tickorigin,
    double tickspace,
    int nminortickspermajor,
    double minorticklength,
    double majorticklength,
    int tickdir
);
```


<i>intercept</i>	Sets the intercept of this axis with the perpendicular axis in physical coordinates.
<i>tickorigin</i>	The tick marks start at this value.
<i>tickspace</i>	Specifies the spacing between minor tick marks.
<i>ntickspermajor</i>	Specifies the number of minor tick marks per major tick mark.
<i>minorticklength</i>	The length of minor tick marks, in .Net device coordinates.
<i>majorticklength</i>	The length of major tick marks, in .Net device coordinates.
<i>tickdir</i>	The direction of the tick marks. Use one of the tick mark direction constants: <code>AXIS_MIN</code> , <code>AXIS_CENTER</code> , or <code>AXIS_MAX</code> .

Use the **SetLineWidth**, **SetLineStyle** and **SetColor** methods to customize the drawing properties of the lines used to draw the axis line and tick marks.

Simple linear axis example

[C#]

```
// Define the coordinate system
double xmin = -5;
double xmax = 15;
double ymin = 0;
double ymax = 105;
CartesianCoordinates simpleScale =
    new CartesianCoordinates(xmin, ymin, xmax, ymax);

// Create the x- and y-axes
LinearAxis xAxis = new LinearAxis(simpleScale, ChartObj.X_AXIS);
LinearAxis yAxis = new LinearAxis(simpleScale, ChartObj.Y_AXIS);

// Create the ChartView object to place graph objects in.
ChartView chartVu = new ChartView();

// Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis);
chartVu.AddChartObject(yAxis);
```

[Visual Basic]

```

` Define the coordinate system
Dim xMin As Double = -5
Dim xMax As Double = 15
Dim yMin As Double = 0
Dim yMax As Double = 15
Dim simpleScale As CartesianCoordinates = _
    New CartesianCoordinates(xMin, yMin, xMax, yMax)

` Create the x- and y-axes
Dim xAxis As LinearAxis = New LinearAxis(simpleScale, ChartObj.X_AXIS)
Dim yAxis As LinearAxis = New LinearAxis(simpleScale, ChartObj.Y_AXIS)

` chartVu Create the ChartView object to place graph objects in.
Dim chartVu As ChartView = New ChartView()

` Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis)
chartVu.AddChartObject(yAxis)

```

Customize the axis by adding the following lines after the creation of the *xAxis* object:

Custom linear axis example

[C#]

```

double xAxisIntercept = -5;
double xAxisIntercept2 = 0;
double xAxisOrigin = 0.0;
double xAxisMinorTickSpace = 1.0;
int xAxisMinorTicksPerMajor = 5;
double xAxisMinorTickLength = 5;
double xAxisMajorTickLength = 10;
int xAxisTickDirection = ChartObj.AXIS_MIN;

xAxis.SetAxisIntercept(xAxisIntercept); // sets the y-intercept of the x-axis
xAxis.SetAxisIntercept2(xAxisIntercept2); // sets the z-intercept of the x-axis
xAxis.SetAxisTicks(xAxisOrigin, xAxisMinorTickSpace,
    xAxisMinorTicksPerMajor, xAxisMinorTickLength,
    xAxisMajorTickLength, xAxisTickDirection);

```

[Visual Basic]

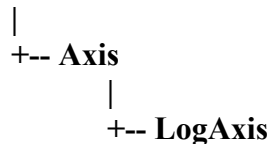
```
Dim xAxisIntercept As Double = -5
Dim xAxisIntercept2 As Double = 0;
Dim xAxisOrigin As Double = 0.0
Dim xAxisMinorTickSpace As Double = 1.0
Dim xAxisMinorTicksPerMajor As Integer = 5
Dim xAxisMinorTickLength As Double = 5
Dim xAxisMajorTickLength As Double = 10
Dim xAxisTickDirection As Integer = ChartObj.AXIS_MIN
Dim xAxis As LinearAxis = New LinearAxis()
xAxis.SetAxisIntercept(xAxisIntercept)
xAxis.SetAxisIntercept2(xAxisIntercept2)
xAxis.SetAxisTicks(xAxisOrigin, xAxisMinorTickSpace, _
    xAxisMinorTicksPerMajor, xAxisMinorTickLength, _
    xAxisMajorTickLength, xAxisTickDirection)
```

Logarithmic Axes

Scientific, engineering and financial applications often require the use of logarithmic axes. Logarithmic axes are useful for the display of data that either has a wide dynamic range and/or data that is exponential in nature. Two common examples that use logarithmic scales are hi-fi speaker charts (db vs. log frequency) and stock market charts.

Class LogAxis

GraphObj



The **LogAxis** class is a concrete subclass of the **Axis** class. Use the **LogAxis** class to create a logarithmic axis with logarithmic spacing between the major tick marks (1, 10, 100...), and linear spacing (2, 3, 4, 5...) between the minor tick marks.

Logarithmic Axis Minimum and Maximum

The minimum and maximum values for a logarithmic axis can have any positive value, as long as the maximum is greater than the minimum. Create an inverted axis by first defining an inverted physical coordinate system using one of the **PhysicalCoordinates** derived classes. The axis minimum and maximum do not have to fall on decade intervals, i.e. 0.1 to 10,000 and can assume any positive range, i.e. 0.23 to 13,100 is valid.

Logarithmic Minor and Major Tick Mark Intervals

The major tick marks for a logarithmic axis use an exponential interval. The exponential interval in physical coordinates transforms to a linear interval in the working coordinate system. Below are examples of the major tick mark locations for a logarithmic axis.

Axis Minimum and Maximum	Axis Major Tick Mark Locations
0.1 to 100.0	0.1, 1.0, 10.0, 100.0
20 to 50,000	20, 200, 2000, 20000
10^{-4} to 1.0	10^{-4} , 10^{-3} , 10^{-2} , 10^{-1} , 1.0

The minor tick marks for a logarithmic axis use a linear interval between the tick marks. For example, a major tick mark interval has endpoints of 10 to 100, a logarithmic interval. The minor ticks in-between the 10 and the 100 use a linear interval of 10 and fall at 20, 30, 40, 50, 60, 70, 80, and 90. For the next major tick mark interval, 100 to 1000, the minor tick mark interval becomes 100 and minor tick marks fall at 200, 300, 400, 500, 600, 700, 800, and 900. The minor tick mark intervals are set equal to the value of the preceding major tick mark interval. If the major tick mark interval uses a non-decade range, for example 3, 30, 300, 30000, the minor tick marks will track the major tick marks. The major tick mark interval of 3 to 30 will use a minor tick mark range of 3, with minor tick marks at 6, 9, 12, 15, 18, 21, 24, and 27.

Logarithmic Axis Intercept

A logarithmic axis has two intercept values, the same as a linear axis.

A logarithmic axis resides in a 3-dimensional physical coordinate system. The minimum and maximum values for the axis provide coordinate information for only one dimension; x-coordinates in the case of an x-axis, and y-coordinates in the case of the y-axis. The missing coordinates needed to position the axis are the axis intercepts. The axis intercepts specifies the y- and z-coordinate position for the x-axis, the x- and z-coordinate position for the y-axis, and the x- and y-coordinate position for the z-axis.

Since the intercept value is specified using the scale of the perpendicular axis, if the perpendicular axis is linear, as in the case of semi-log graphs, the intercept value can be positive, negative, or 0.0. If the perpendicular axis is logarithmic, the intercept value is restricted to a positive range.

Logarithmic Axis Tick Mark Origin

The starting value for the major tick marks does not need to fall at the end of the axis range. For example, the axis may have a range of 0.175 to 195. It would not make sense to start the major tick mark placement at 0.175. The major tick marks would end up placed at 0.175, 1.75, 17.5 and 175. The minor tick marks would make even less sense. A

better major tick mark placement is 0.2, 2, 20, and 200. The minor tick marks will also fall on even values.

The logarithmic axis tick mark origin controls the placement of the first major tick mark. The other major and minor tick mark positions are automatically calculated based on the initial position of the first major tick mark.

The tick mark origin must reside in the bounds defined by the axis minimum and maximum, inclusive of the endpoints. It does not need to be near an endpoint however.

LogAxis Constructors

There are two constructors for **LogAxis** objects.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal axtype As Integer _
)
```

```
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal axtype As Integer, _
    ByVal minval As Double, _
    ByVal maxval As Double _
)
```

```
[C#]
public LogAxis(
    PhysicalCoordinates transform,
    int axtype
);
```

```
public LogAxis(
    PhysicalCoordinates transform,
    int axtype,
    double minval,
    double maxval
);
```

transform Places the axes in the coordinate system defined by transform.

axtype Specifies if the axis is an x-axis (X_AXIS), y-axis (Y_AXIS) or a z-axis (Z_AXIS).

minval Sets the minimum value for the axis.

maxval Sets the maximum value for the axis.

The first **LogAxis** constructor assumes that the axis extents match the extents of the underlying coordinate system, *transform*. The second **LogAxis** constructor sets the axis extents to the specified minimum and maximum values, regardless of the underlying coordinate system.

Other axis properties: axis intercept, tick mark lengths, tick mark direction and axis tick mark origin are automatically calculated using an auto-axis method. These properties can be explicitly set if you need to override the automatically calculated values.

SetAxisIntercept and SetAxisIntercept2 methods

The **SetAxisIntercept** and **SetAxisIntercept2** methods set the two intercepts needed to position an axis. The coordinates they apply to depends on the axis.

X_AXIS **SetAxisIntercept** sets the y-intercept of the x-axis. **SetAxisIntercept2** sets the z-intercept of the x-axis.

Y_AXIS **SetAxisIntercept** sets the x-intercept of the y-axis. **SetAxisIntercept2** sets the z-intercept of the y-axis.

Z_AXIS **SetAxisIntercept** sets the x-intercept of the z-axis. **SetAxisIntercept2** sets the y-intercept of the z-axis.

```
[Visual Basic]
Public Sub SetAxisIntercept( _
    ByVal intercept As Double _
)
```

```
Public Sub SetAxisIntercept2( _
    ByVal intercept As Double _
)
```

```
[C#]
public void SetAxisIntercept(
    double intercept
);
```

```
public void SetAxisIntercept2(
    double intercept
);
```

SetAxisTicks method

```
[Visual Basic]
Overloads Public Sub SetAxisTicks( _
    ByVal tickorigin As Double, _
    ByVal nlogtickformat As Integer _
)
```

```
[Visual Basic]
Overloads Public Sub SetAxisTicks( _
    ByVal origin As Double, _
    ByVal nlogtickformat As Integer, _
    ByVal minorticklength As Double, _
    ByVal majorticklength As Double, _
)
```

```

        ByVal tickdir As Integer _
    )

    [C#]
    public void SetAxisTicks(
        double tickorigin,
        int nlogtickformat
    );

    public void SetAxisTicks(
        double origin,
        int nlogtickformat,
        double minorticklength,
        double majorticklength,
        int tickdir
    );

```

intercept

Sets the intercept of this axis with the perpendicular axis in physical coordinates.

nlogtickformat

This parameter specifies which minor tick marks are flagged for labels. Logarithmic axis minor tick mark labels can become very crowded. It is possible to choose values that may overlap or not display. Valid *nlogtickformat* values are:

- 0 No minor tick mark labels
- 1 Place a label at tick mark 0 in each decade.
- 2 Place a label at minor tick marks 1, 3 and 5 in each decade.
- 3 Place a label at minor tick marks 0, 1, 2, 3 and 5 in each decade.
- 4 Place a label at minor tick marks 0, 1, 2, 3, 4 and 5 in each decade.
- 5 Place a label at minor tick marks 0, 1, 2, 3, 4, 5 and 6 in each decade.
- 6 Place a label at minor tick marks 0, 1, 2, 3, 4, 5, 6 and 7 in each decade.
- 7 Place a label at minor tick marks 0, 1, 2, 3, 4, 5, 6, 7 and 8 in each decade.
- 8 Place a label at minor tick marks 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 in each decade.

minorticklength

The length of minor tick marks, in .Net device coordinates.

<i>majorticklength</i>	The length of major tick marks, in .Net device coordinates.
<i>ticdir</i>	The direction of the tick marks. Use one of the tick mark direction constants: <code>AXIS_MIN</code> , <code>AXIS_CENTER</code> , or <code>AXIS_MAX</code> .

The **SetLineWidth**, **SetLineStyle** and **SetColor** methods are used to customize the drawing properties of the lines used to draw the axis line and tick marks.

Simple log axis example

[C#]

```
double xmin = 0;
double xmax = 1000;
double ymin = 0.2;
double ymax = 2000;
CartesianCoordinates logYScale =
    new CartesianCoordinates(ChartObj.LINEAR_SCALE, ChartObj.LOG_SCALE);
logYScale.SetCoordinateBounds(xmin, ymin, xmax, ymax);

// Create a linear x-axis and a logarithmic y-axis
LinearAxis xAxis = new LinearAxis (logYScale, ChartObj.X_AXIS);
LogAxis yAxis = new LogAxis (logYScale, ChartObj.Y_AXIS);

// Create the ChartView object to place graph objects in.
ChartView chartVu = new ChartView();

// Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis);
chartVu.AddChartObject(yAxis);
```

[Visual Basic]

```
Dim xmin As Double = 0
Dim xmax As Double = 1000
Dim ymin As Double = 0.2
Dim ymax As Double = 2000
Dim logYScale As CartesianCoordinates = _
    New CartesianCoordinates(ChartObj.LINEAR_SCALE, ChartObj.LOG_SCALE)
logYScale.SetCoordinateBounds(xmin, ymin, xmax, ymax)

' Create a linear x-axis and a logarithmic y-axis
Dim xAxis As LinearAxis = New LinearAxis(logYScale, ChartObj.X_AXIS)
Dim yAxis As LogAxis = New LogAxis(logYScale, ChartObj.Y_AXIS)
```



```
' Create the ChartView object to place graph objects in.
Dim chartVu As ChartView = New ChartView()

' Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis)
chartVu.AddChartObject(yAxis)
```

Should want to customize the axis you can add the following lines after the *yAxis* object is created:

Custom logarithmic axis example

[C#]

```
// Place the y-axis on the right side of the graph with tick marks
// point towards the right.
double yAxisIntercept = 1000;
double yAxisIntercept2 = 0;

// Major tick marks at 0.2, 2, 20, 200 and 2000
double yAxisOrigin = 0.2;
// In addition to major tick marks, labels flagged for some minor tick marks
int yAxisLogFormat = 1;
double yAxisMinorTickLength = 5;
double yAxisMajorTickLength = 10;
int yAxisTickDirection = ChartObj.AXIS_MAX;

yAxis.SetAxisIntercept(yAxisIntercept);
yAxis.SetAxisIntercept2(yAxisIntercept2);
yAxis.SetAxisTicks(yAxisOrigin, yAxisLogFormat, yAxisMinorTickLength,
                  yAxisMajorTickLength, yAxisTickDirection);
```

[Visual Basic]

```
' Place the y-axis on the right side of the graph with tick marks
' point towards the right.
Dim yAxisIntercept As Double = 1000
' yAxisOrigin Major tick marks at 0.2, 2, 20, 200 and 2000
Dim yAxisOrigin As Double = 0.2
' In addition to major tick marks, labels flagged for some minor tick marks
Dim yAxisLogFormat As Integer = 1
Dim yAxisMinorTickLength As Double = 5
```

```

Dim yAxisMajorTickLength As Double = 10
Dim yAxisTickDirection As Integer = ChartObj.AXIS_MAX

yAxis.SetAxisIntercept(yAxisIntercept)
yAxis.SetAxisTicks(yAxisOrigin, yAxisLogFormat, yAxisMinorTickLength, _
                  yAxisMajorTickLength, yAxisTickDirection)

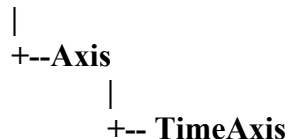
```

Date/Time Axes

The date/time axis is used in combination with a **TimeCoordinates** physical coordinate system. The axis major and minor tick marks correspond to the common date/time divisions of second, minute, hour, day, week, month and year. The date/time axes supported with this software are very complex, because they take into account the varying number of days in months and years. The axes also take into account non-continuous date/time scales where a 5-day week is used, or where a full day consists of a specific time interval that can be something less than a 24-hour day.

Class TimeAxis

GraphObj



The **TimeAxis** class creates an axis with date/time specific spacing between minor and major tick marks, not necessarily uniform as with a **LinearAxis**. The **TimeAxis** extends the **Axis** class.

Date/Time Axis Minimum and Maximum

The minimum and maximum values for a date/time axis can have any valid date/time value, specified using the class **ChartCalendar**. The axis maximum value should be later in time than the minimum. Create an inverted axis by first defining an inverted physical coordinate system using the **TimeCoordinates** class. The axis minimum and maximum do not have to fall on even time or date intervals and can assume any date compatible with the **ChartCalendar** class. For example:

Starting Date and Time	Ending Date and Time	Range
1/1/1972 00:00:00	1/1/1999 00:00:00	27 years

11/04/1997 8:30:00	11/04/1997 16:00:00	7 hours 30 minutes
11/28/2000 8:31:22	1/14/2001 15:14:33	48 days 6 hours 43 minutes 11 sec

Date/Time Minor and Major Tick Mark Intervals

The predefined date/time axis tick mark constants listed below specify both major and minor tick mark spacing.

Date/Time Axis Tick Mark Constants	Description
TIMEAXIS_50YEAR10YEAR	50 year major tick mark spacing, 10 year minor tick mark spacing
TIMEAXIS_20YEAR5YEAR	20 year major tick mark spacing, 5 year minor tick mark spacing
TIMEAXIS_10YEARYEAR	10 year major tick mark spacing, 1 year minor tick mark spacing
TIMEAXIS_5YEARYEAR	5 year major tick mark spacing, 1 year minor tick mark spacing
TIMEAXIS_YEAR	1 year major tick mark spacing
TIMEAXIS_YEARQUARTER	1 year major tick mark spacing, 1 quarter minor tick mark spacing
TIMEAXIS_YEARMONTH	1 year major tick mark spacing, 1 month minor tick mark spacing
TIMEAXIS_QUARTER	1 quarter major tick mark spacing
TIMEAXIS_QUARTERMONTH	1 quarter major tick mark spacing, 1 month minor tick mark spacing
TIMEAXIS_MONTH	1 month major tick mark spacing
TIMEAXIS_MONTHWEEK	1 month major tick mark spacing, 1 week minor tick mark spacing
TIMEAXIS_MONTHDAY	1 month major tick mark spacing, 1 day minor tick mark spacing

TIMEAXIS_WEEK	1 week major tick mark spacing
TIMEAXIS_WEEKDAY	1 week major tick mark spacing, 1 day minor tick mark spacing
TIMEAXIS_DAY	1 day major tick mark spacing
TIMEAXIS_DAY12HOUR	1 day major tick mark spacing, 12 hour minor tick mark spacing
TIMEAXIS_DAY8HOUR	1 day major tick mark spacing, 8 hour minor tick mark spacing
TIMEAXIS_DAY4HOUR	1 day major tick mark spacing, 4 hour minor tick mark spacing
TIMEAXIS_DAY2HOUR	1 day major tick mark spacing, 2 hour minor tick mark spacing
TIMEAXIS_DAYHOUR	1 day major tick mark spacing, 1 hour minor tick mark spacing
TIMEAXIS_12HOURHOUR	12 hour major tick mark spacing, 1 hour minor tick mark spacing
TIMEAXIS_8HOURHOUR	8 hour major tick mark spacing, 1 hour minor tick mark spacing
TIMEAXIS_4HOURHOUR	4 hour major tick mark spacing, 1 hour minor tick mark spacing
TIMEAXIS_2HOURHOUR	2 hour major tick mark spacing, 1 hour minor tick mark spacing
TIMEAXIS_HOUR	1 hour major tick mark spacing
TIMEAXIS_HOUR30MINUTE	1 hour major tick mark spacing, 30 minute minor tick mark spacing
TIMEAXIS_HOUR15MINUTE	1 hour major tick mark spacing, 15 minute minor tick mark spacing
TIMEAXIS_HOUR10MINUTE	1 hour major tick mark spacing, 10 minute minor tick mark spacing
TIMEAXIS_HOUR5MINUTE	1 hour major tick mark spacing, 5 minute minor tick mark spacing

TIMEAXIS_HOUR2MINUTE	1 hour major tick mark spacing, 2 minute minor tick mark spacing
TIMEAXIS_HOURMINUTE	1 hour major tick mark spacing, 1 minute minor tick mark spacing
TIMEAXIS_30MINUTEMINUTE	30 minute major tick mark spacing, 1 minute minor tick mark spacing
TIMEAXIS_15MINUTEMINUTE	15 minute major tick mark spacing, 1 minute minor tick mark spacing
TIMEAXIS_10MINUTEMINUTE	10 minute major tick mark spacing, 1 minute minor tick mark spacing
TIMEAXIS_5MINUTEMINUTE	5 minute major tick mark spacing, 1 minute minor tick mark spacing
TIMEAXIS_2MINUTEMINUTE	2 minute major tick mark spacing, 1 minute minor tick mark spacing
TIMEAXIS_MINUTE	1 minute major tick mark spacing
TIMEAXIS_MINUTE30SECOND	1 minute major tick mark spacing, 30 second minor tick mark spacing
TIMEAXIS_MINUTE15SECOND	1 minute major tick mark spacing, 15 second minor tick mark spacing
TIMEAXIS_MINUTE10SECOND	1 minute major tick mark spacing, 10 second minor tick mark spacing
TIMEAXIS_MINUTE5SECOND	1 minute major tick mark spacing, 5 second minor tick mark spacing
TIMEAXIS_MINUTE2SECOND	1 minute major tick mark spacing, 2 second minor tick mark spacing
TIMEAXIS_MINUTESECOND	1 minute major tick mark spacing, 1 second minor tick mark spacing
TIMEAXIS_30SECONDSECOND	30 second major tick mark spacing, 1 second minor tick mark spacing
TIMEAXIS_15SECONDSECOND	15 second major tick mark spacing, 1 second minor tick mark spacing
TIMEAXIS_10SECONDSECOND	10 second major tick mark spacing, 1 second minor tick mark spacing

TIMEAXIS_5SECONDSECOND	5 second major tick mark spacing, 1 second minor tick mark spacing
TIMEAXIS_2SECONDSECOND	2 second major tick mark spacing, 1 second minor tick mark spacing
TIMEAXIS_SECOND	1 second major tick mark spacing

Sunday is the first day of the week for 7-day weeks, while Monday is the first day of the week for 5-day weeks.

It may not be immediately obvious, but the major tick marks for date/time scales do not necessarily fall on equal intervals. If the tick marks are set to the TIMEAXIS_MONTHDAY value, there may be 28, 29, 30 or 31 days between each months major tick mark. In most cases, the major tick mark coincides with a minor tick mark. For example, if the TIMEAXIS_MONTHDAY setting is used, the major tick mark for a month falls at the first day of the month, coinciding with the minor tick mark for that day. If the TIMEAXIS_WEEKDAY setting is used, the major tick mark for a WEEK falls at the first day of the week, coinciding with the minor tick mark for that day. Situations where the major and minor tick marks do not coincide involve weeks as minor tick marks. If the TIMEAXIS_MONTHWEEK setting is used, the first day of the month may or may not correspond to the first day of the week (Sunday or Monday). Major tick marks fall on the first day of each month, and minor tick marks fall on the first day of each week.

Combine these irregularities with a 5- or 7-day workweek option, and the non-24 hour day option and you can see that a generalized algorithm to define the positions of tick marks is a difficult task. For example: you are a stock trader and you want to track the price/volume characteristics of a stock from the last 5 minutes of the regular trading day, to the first 5 minutes of the next regular trading day, specifically from 3:55 PM Friday, Sept 29, 2000 to 9:35 AM Monday, Oct 2, 2000. The time range under consideration is only 10 minutes, since the market closes Friday at 4:00 PM and opens Monday at 9:30 PM. The resulting axis should reflect this 10-minute range, even though the actual time elapsed is 3,930 minutes. You should be able to specify the axis minimum and maximum values using the actual dates and times. You also need to set a 5-day workweek mode and establish a day that has a time range of 9:30 AM to 4:00 PM.

SetAxisIntercept and SetAxisIntercept2 methods

A date/time axis has an intercept value, the same as a linear axis. An axis resides in a 3-dimensional physical coordinate system. The minimum and maximum values for the axis provide coordinate information for only one dimension; x-coordinates in the case of an x-axis, and y-coordinates in the case of the y-axis. The missing coordinates needed to position the axis are the axis intercepts. The axis intercepts specifies the y- and z-coordinate position for the x-axis, the x- and z-coordinate position for the y-axis and the x- and y-coordinate position for the z-axis.

There are three main constructors for **TimeAxis** objects. The first two **TimeAxis** constructors assume that the axis extents match the extents of the underlying coordinate system, *transform*. The third **TimeAxis** constructor sets the axis extents to the specified minimum and maximum values, regardless of the underlying coordinate system.

TimeAxis constructors

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As TimeCoordinates _
)

Overloads Public Sub New( _
    ByVal transform As TimeCoordinates, _
    ByVal ntickmarkbase As Integer _
)

Overloads Public Sub New( _
    ByVal transform As TimeCoordinates, _
    ByVal dstart As ChartCalendar, _
    ByVal dstop As ChartCalendar _
)

[C#]
public TimeAxis(
    TimeCoordinates transform
);

public TimeAxis(
    TimeCoordinates transform,
    int ntickmarkbase
);

public TimeAxis(
    TimeCoordinates transform,
    ChartCalendar dstart,
    ChartCalendar dstop
);
```

<i>transform</i>	The time coordinate system the axis is placed in. If the starting and ending dates of the axis are not explicitly set, the axis uses the starting and ending dates of the <i>transform</i> time-scale.
<i>dstart</i>	The starting date value for the axis.
<i>dstop</i>	The ending date value for the axis
<i>ntickmarkbase</i>	This field defines the major and minor tick mark spacing for a time axis. Use one of the Date/time axis tick mark mode constants: TIMEAXIS_YEARMONTH, TIMEAXIS_DAYHOUR for example.

Other axis properties: axis intercept, tick mark lengths, tick mark direction are automatically calculated using an auto-axis method. These properties can be explicitly set if you need to override the automatically calculated values.

SetAxisIntercept and SetAxisIntercept2 methods

The **SetAxisIntercept** and **SetAxisIntercept2** methods set the two intercepts needed to position an axis. The coordinates they apply to depends on the axis.

X_AXIS	SetAxisIntercept sets the y-intercept of the x-axis. SetAxisIntercept2 sets the z-intercept of the x-axis.
Y_AXIS	SetAxisIntercept sets the x-intercept of the y-axis. SetAxisIntercept2 sets the z-intercept of the y-axis.
Z_AXIS	SetAxisIntercept sets the x-intercept of the z-axis. SetAxisIntercept2 sets the y-intercept of the z-axis.

```
[Visual Basic]
Public Sub SetAxisIntercept( _
    ByVal intercept As Double _
)

Public Sub SetAxisIntercept2 _
    ByVal intercept As Double _
)

[C#]
public void SetAxisIntercept(
    double intercept
);

public void SetAxisIntercept2
    double intercept
);
```

SetAxisTicksAttributes method

```
[Visual Basic]
Public Sub SetAxisTicksAttributes( _
    ByVal minorticklength As Double, _
    ByVal majorticklength As Double, _
    ByVal tickdir As Integer _
)

[C#]
public void SetAxisTicksAttributes(
    double minorticklength,
    double majorticklength,
    int tickdir
);
```

intercept Sets the intercept of this axis with the perpendicular axis in physical coordinates.

<i>minorticklength</i>	Specifies the length of a minor tick mark in .Net device coordinates.
<i>majorticklength</i>	Specifies the length of a major tick mark in .Net device coordinates.
<i>tickdir</i>	Specifies the direction of the tick marks with respect to axis line. Use one of the following tick direction constants: <code>AXIS_MIN</code> , <code>AXIS_CENTER</code> , <code>AXIS_MAX</code> .

Customize the line and tick mark drawing properties of the axis using the **SetLineWidth**, **SetLineStyle** and **SetColor** methods.

Simple time axis example

[C#]

```
// Define a Time coordinate system
ChartCalendar xMin = new ChartCalendar(1996, ChartObj.FEBRUARY, 5);
ChartCalendar xMax = new ChartCalendar(2002, ChartObj.JANUARY, 5);
double yMin = 0;
double yMax = 105;

TimeCoordinates simpleTimeScale;
simpleTimeScale = new TimeCoordinates(xMin, yMin, xMax, yMax);
// Create the time axis (x-axis is assumed)
TimeAxis xAxis = new TimeAxis(simpleTimeScale);
// Create the linear y-axis
LinearAxis yAxis = new LinearAxis(simpleTimeScale, ChartObj.Y_AXIS);
```

[Visual Basic]

```
' Define a Time coordinate system
Dim xMin As ChartCalendar = New ChartCalendar(1996, ChartObj.FEBRUARY, 5)
Dim xMax As ChartCalendar = New ChartCalendar(2002, ChartObj.JANUARY, 5)
Dim yMin As Double = 0
Dim yMax As Double = 105

Dim simpleTimeScale As TimeCoordinates
simpleTimeScale = New TimeCoordinates(xMin, yMin, xMax, yMax)
' Create the time axis (x-axis is assumed)
Dim xAxis As TimeAxis = New TimeAxis(simpleTimeScale)
' Create the linear y-axis
Dim yAxis As LinearAxis = New LinearAxis(simpleTimeScale, ChartObj.Y_AXIS)
```

```
' Create the ChartView object to place graph objects in.
Dim chartVu As ChartView = New ChartView()
```

```
' Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis)
chartVu.AddChartObject(yAxis)
```

Custom time axis example

[C#]

```
// Define a Time coordinate system
ChartCalendar xMin = new ChartCalendar(1996, ChartObj.FEBRUARY, 5);
ChartCalendar xMax = new ChartCalendar(2002, ChartObj.JANUARY, 5);
double axisintercept = 0.0;
double axisintercept2 = 0.0;
double yMin = 0;
double yMax = 105;
int xAxisTickMarkFormat = ChartObj.TIMEAXIS_MONTHWEEK;
double xAxisMinorTickLength = 5;
double xAxisMajorTickLength = 10;
int xAxisTickDirection = ChartObj.AXIS_MIN;
TimeCoordinates simpleTimeScale;
simpleTimeScale = new TimeCoordinates(xMin, yMin, xMax, yMax);

// Create the time axis (x-axis is assumed)
TimeAxis xAxis = new TimeAxis(simpleTimeScale, xAxisTickMarkFormat);
xAxis.SetAxisIntercept(axisintercept);
xAxis.SetAxisIntercept2(axisintercept2);
xAxis.SetAxisTicksAttributes(xAxisMinorTickLength,
                             xAxisMajorTickLength, xAxisTickDirection);

// Create the linear y-axis
LinearAxis yAxis =
new LinearAxis(simpleTimeScale, ChartObj.Y_AXIS);

// Create the ChartView object to place graph objects in.
ChartView chartVu = new ChartView();

// Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis);
chartVu.AddChartObject(yAxis);
```

[Visual Basic]

```

' Define a Time coordinate system
Dim xMin As ChartCalendar = New ChartCalendar(1996, ChartObj.FEBRUARY, 5)
Dim xMax As ChartCalendar = New ChartCalendar(2002, ChartObj.JANUARY, 5)
Dim yMin As Double = 0
Dim yMax As Double = 105
Dim xAxisTickMarkFormat As Integer = ChartObj.TIMEAXIS_MONTHWEEK
Dim xAxisMinorTickLength As Double = 5
Dim xAxisMajorTickLength As Double = 10
Dim xAxisTickDirection As Integer = ChartObj.AXIS_MIN
Dim simpleTimeScale As TimeCoordinates
simpleTimeScale = New TimeCoordinates(xMin, yMin, xMax, yMax)

' Create the time axis (x-axis is assumed)
Dim xAxis As TimeAxis = New TimeAxis(simpleTimeScale, xAxisTickMarkFormat)
xAxis.SetAxisTicksAttributes(xAxisMinorTickLength, _
                             xAxisMajorTickLength, xAxisTickDirection)

' Create the linear y-axis
Dim yAxis As LinearAxis = New LinearAxis(simpleTimeScale, ChartObj.Y_AXIS)

' Create the ChartView object to place graph objects in.
Dim chartVu As ChartView = New ChartView()

' Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis)
chartVu.AddChartObject(yAxis)

```


8. Axis Labels

AxisLabels

NumericAxisLabels

TimeAxisLabels

StringAxisLabels

Axis Labels

Axis labels are numeric or text strings placed next to axis tick marks, indicating the scale of the axis. Axis labels are a separate class from the axis classes. An axis class, i.e. any class derived from **Axis**, can exist independent of axis labels. Many graphs use axes that do not have labels. For example, the y-axis on the left side of a graph may have labels, while an identical axis on the right hand side of the graph may not. The axis label classes require a valid reference axis class and cannot exist independently.

There are five axis labels classes: the **AxisLabels** abstract base class and concrete subclasses **NumericAxisLabels**, **TimeAxisLabels**, and **StringAxisLabels**..

Label Formats

An axis label can take many forms. The various axis label formats are divided between the axis labels classes in the following manner.

NumericAxisLabels

The **LinearAxis** and **LogAxis** axis types use this class.

- Full decimal conversion (0.000015)
- Scientific notation (1.5e-5)
- Exponent notation (1.5×10^{-5})
- Percent format (76%)
- Business format where B, M and K are used to represent billions, millions and thousands (1043M, 11.0K)
- Currency format (\$123432)
- Business currency format – The business format combined with the currency format (\$123K)

StringAxisLabels

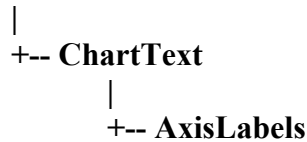
The **LinearAxis** and **LogAxis** axis types use this class.

Arbitrary strings (“MA”, “PA”, “JEFF”, ”Sales”)

TimeAxisLabels

The **TimeAxis** axis types use this class.

- Time formats (hh:mm:ss, hh:mm, mm:ss, etc.)
- Date formats (mm/dd/yy, dd/mm/yy, mm/yy, etc.)
- Time/Date formats (mmm/ddd/yyyy hh:mm:ss)

Class AxisLabels**GraphObj**

The **AxisLabels** class is the abstract base class for all axis label objects. It contains the properties and methods common to subclasses implementing more specialized axis labels.

Axis Label Text

The **AxisLabels** class includes a reference to a **Font** object. If a valid font is not supplied a default font is created and used. Every axis labels object can have a unique font associated with it. The **Font** object defines the font typeface, size, and style. The font for any of the axis labels can be set using the **AxisLabels.SetTextFont** method. The **AxisLabels** class manages other text attributes not directly associated with the font.

These include the text foreground color, the text background color and the rotation of the text if it is different from the normal horizontal orientation. It is common to rotate x-axis labels 90 degrees, so that they are vertical rather than horizontal, in order to squeeze more tick mark labels in along the x-axis.

Axis Labels Positioning

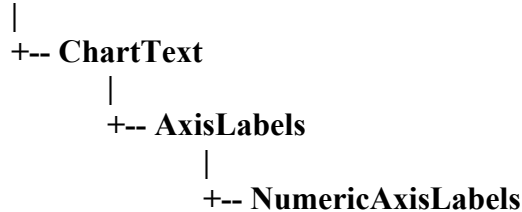
The **AxisLabels** class manages the placement of the axis labels with respect to the underlying axis tick marks. Labels can be placed above or below the tick marks of a horizontal x-axis, and to the left or right of the tick marks for a vertical y-axis. The axis label justification constants **AXIS_MIN** and **AXIS_MAX** are used for this purpose. The **AXIS_MIN** constant places the text label on the side of the tick mark that is in the direction of the perpendicular axis coordinate system minimum. The **AXIS_MAX** is

much the same, except that it places the label on the side that is in the direction of the perpendicular axis coordinate system maximum. Axis labels should not actually touch the tick marks, so x and y offsets are factored in. The programmer can modify these offsets.

Numeric Axis Labels

Class NumericAxisLabels

GraphObj



The **NumericAxisLabels** class extends the **AxisLabels** class, adding extensive numeric formatting capability. It labels axes created using the **LinearAxis** and **LogAxis** classes.

Label formats

An axis label can take many forms. Variations on these forms include:

- Full decimal conversion (0.000015)
- Scientific notation (1.5e-5)
- Exponent notation (1.5x10⁻⁵)
- Percent format (76%)
- Business format where B, M and K are used to represent billions, millions and thousands (1043M, 11.0K)
- Currency format (\$123432)
- Business currency format – The business format combined with the currency format (\$123K)

Depending on the scaling of the associated axis, the numeric values of the axis labels may be very large or very small numbers requiring a great deal of space to display. Various techniques are used to abbreviate the numeric value, reducing the space requirements. Expressing a numeric value in scientific notation can reduce the amount of space a label requires, if the numeric value requires eight or more digits. If the label values end in a lot of zeros (10000000, 9000000, 8000000...), a major reduction in space is achieved by using the business format which replaces all of the zeros with a letter (10M, 9M, 8M, ...).

The axis numeric labels constants are listed below:

Numeric Format Constant Description

DECIMALFORMAT	Decimal format, i.e. 1234.563
SCIENTIFICFORMAT	Scientific or exponential format: 1.23e3
BUSINESSFORMAT	Business format where the letters K, M, B or T are appended on the end a truncated numeric value, i.e. 1.23, 14K, 44M, 32B, 3.0T
ENGINEERINGFORMAT	If the absolute value of the label is greater than 1.0e6, or less than 1.0e-6, the scientific format is used, else the decimal format is used.
PERCENTFORMAT	The value of a label is multiplied by 100 and the character ‘%’ is appended on the end of the label.
EXPONENTFORMAT	The value of a label is multiplied by 100 and the character ‘%’ is appended on the end of the label.
CURRENCYBUSINESSFORMAT	A ‘\$’ character is appended to the front of the label and the values are abbreviated the same as the BUSINESSFORMAT
CURRENCYFORMAT	A ‘\$’ character is appended to the front of the label. .

NumericAxisLabels constructor

There is only one main constructor for **NumericAxisLabels** objects.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal baseaxis As Axis _
)

[C#]
public NumericAxisLabels(
    Axis baseaxis
);
```

baseaxis This is the axis the axis labels are for.

Other axis label properties: font, rotation, numeric format, axis labels direction and numeric precision are automatically set. These properties can be explicitly set if you need to override the automatically calculated values.

SetAxisLabels method

```
[Visual Basic]
Overloads Public Sub SetAxisLabels( _
    ByVal font As Font, _
    ByVal rotation As Double, _
    ByVal labdir As Integer, _
    ByVal decimalpos As Integer, _
    ByVal labelends As Integer, _
    ByVal labcolor As Color _
)

[C#]
public void SetAxisLabels(
    Font font,
    double rotation,
    int labdir,
    int decimalpos,
    int labelends,
    Color labcolor
);
```

SetAxisLabelsFormat method

```
[Visual Basic]
Public Sub SetAxisLabelsFormat( _
    ByVal format As Integer _
)

[C#]
public void SetAxisLabelsFormat(
    int format
);
```

<i>font</i>	The font object used to display the axis label text.
<i>rotation</i>	The rotation, in degrees, of label text in the normal viewing plane.
<i>labdir</i>	The justification of the axis label (AXIS_MIN or AXIS_MAX) with respect to the tick mark endpoint.
<i>decimal</i>	Sets the number of digits to the right of the decimal point for numeric axis labels.
<i>labelends</i>	Specifies whether there should be labels for the axis minimum (LABEL_MIN), maximum (LABEL_MAX) or tick mark starting point (LABEL_ORIGIN). The value of these constants can be OR'd together. The value of LABEL_MIN LABEL_MAX LABEL_ORIGIN is LABEL_ALL

<i>labcolor</i>	The color of the label text.
<i>format</i>	Sets the numeric format for the axis labels. Use one of the numeric format constants: DECIMALFORMAT, SCIENTIFICFORMAT, EXPONENTFORMAT, BUSINESSFORMAT, ENGINEERINGFORMAT, PERCENTFORMAT, CURRENCYFORMAT, CURRENCYBUSINESSFORMAT.

Simple numeric axis labels example

[C#]

```
// Define the coordinate system
double xmin = -5;
double xmax = 15;
double ymin = 0;
double ymax = 105;
CartesianCoordinates simpleScale =
    new CartesianCoordinates(xmin, ymin, xmax, ymax);

// Create the x- and y-axes
LinearAxis xAxis =
    new LinearAxis(simpleScale, ChartObj.X_AXIS);
LinearAxis yAxis = new LinearAxis(simpleScale, ChartObj.Y_AXIS);
NumericAxisLabels xAxisLabels = new NumericAxisLabels(xAxis);
NumericAxisLabels yAxisLabels = new NumericAxisLabels(yAxis);
// Create the ChartView object to place graph objects in.
ChartView chartVu = new ChartView();

// Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis);
chartVu.AddChartObject(yAxis);
chartVu.AddChartObject(xAxisLabels);
chartVu.AddChartObject(yAxisLabels);
```

[Visual Basic]

```
' Define the coordinate system
Dim xmin As Double = -5
Dim xmax As Double = 15
Dim ymin As Double = 0
Dim ymax As Double = 105
```

```

Dim simpleScale As CartesianCoordinates = _
    New CartesianCoordinates(xMin, yMin, xMax, yMax)

' Create the x- and y-axes
Dim xAxis As LinearAxis = _
    New LinearAxis(simpleScale, ChartObj.X_AXIS)
Dim yAxis As LinearAxis = New LinearAxis(simpleScale, ChartObj.Y_AXIS)
Dim xAxisLabels As NumericAxisLabels = New NumericAxisLabels(xAxis)
Dim yAxisLabels As NumericAxisLabels = New NumericAxisLabels(yAxis)
' Create the ChartView object to place graph objects in.
Dim chartVu As ChartView = New ChartView()
' Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis)
chartVu.AddChartObject(yAxis)
chartVu.AddChartObject(xAxisLabels)
chartVu.AddChartObject(yAxisLabels)

```

Should want to customize the axis you can add the following lines after the *xAxisLabels* object is created:

Custom numeric axis labels example

[C#]

```

Font labelfont = new Font("Helvetica", 10, FontStyle.BOLD);

double xAxisLabelsRotation = 0.0;
int xAxisLabelsDir = ChartObj.AXIS_MIN;
int xAxisLabelsDecimal = 1;
int xAxisLabelsEnds = ChartObj.LABEL_ALL;
Color xAxisLabelsColor = Color.Black;
int xAxisNumericFormat = ChartObj.DECIMALFORMAT;

xAxisLabels.SetAxisLabels( labelfont, xAxisLabelsRotation,
                           xAxisLabelsDir, xAxisLabelsDecimal,
                           xAxisLabelsEnds, xAxisLabelsColor);
xAxisLabels.SetAxisLabelsFormat(xAxisNumericFormat);

```

[Visual Basic]

```

Dim labelfont As Font = New Font("Helvetica", 10, FontStyle.Bold)
Dim xAxisLabelsRotation As Double = 0.0
Dim xAxisLabelsDir As Integer = ChartObj.AXIS_MIN

```

```

Dim xAxisLabelsDecimal As Integer = 1
Dim xAxisLabelsEnds As Integer = ChartObj.LABEL_ALL
Dim xAxisLabelsColor As Color = Color.Black
Dim xAxisNumericFormat As Integer = ChartObj.DECIMALFORMAT

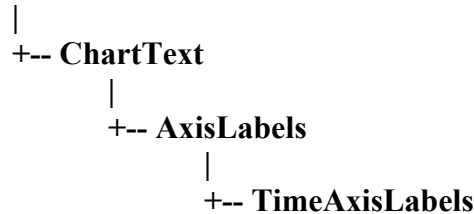
xAxisLabels.SetAxisLabels(labelfont, xAxisLabelsRotation, _
                           xAxisLabelsDir, xAxisLabelsDecimal, _
                           xAxisLabelsEnds, xAxisLabelsColor)
xAxisLabels.SetAxisLabelsFormat(xAxisNumericFormat)

```

Time and Date Axis Labels

Class TimeAxisLabels

GraphObj



The **TimeAxisLabels** class extends the **AxisLabels** class, adding extensive time and date formatting capability. Use it to label axes created using the **TimeAxis** class.

Label formats

A time axis label can take many forms. Variations on these forms include:

Time formats (hh:mm:ss, hh:mm, mm:ss, etc.)

- Date formats (mm/dd/yy, dd/mm/yy, mm/yy, etc.)

There are more ways to format time and date information than numeric data. The **QCChart3D for .Net** software directly supports twelve time formats and eighteen date formats. It is also possible to create custom date/time formats. The software makes use of the **System.DateTime.ToString** method to format times and dates. A table listing predefined date/time formats appears below.

Date/Time Format Constant	Format String	Example String Result
TIMEDATEFORMAT_MSDDD	"mm:ss.fff"	12.33.999
TIMEDATEFORMAT_MSDD	"mm:ss.ff"	12.33.99

TIMEDATEFORMAT_MSD	"mm:ss.f"	12.33.9
TIMEDATEFORMAT_MS	"m:ss"	12:33
TIMEDATEFORMAT_12HMSDD	"h:mm:ss.ff"	11:12:33.99
TIMEDATEFORMAT_12HMSD	"h:mm:ss.f"	11:12:33.9
TIMEDATEFORMAT_12HMS	"h:mm:ss"	11:12:33
TIMEDATEFORMAT_12HM	"h:mm"	11:12
TIMEDATEFORMAT_24HMDDD	"H:mm:ss.ff"	23:12:33.99
TIMEDATEFORMAT_24HMDD	"H:mm:ss.f"	23:12:33.9
TIMEDATEFORMAT_24HMS	"H:mm:ss"	23:12:33
TIMEDATEFORMAT_24HM	"H:mm"	23:12
TIMEDATEFORMAT_STANDARD	"MMMMMM dd, yyyy"	December 7, 2000
TIMEDATEFORMAT_MDY	"M/dd/yy"	12/07/00
TIMEDATEFORMAT_DMY	"d/MM/yy"	7/12/00
TIMEDATEFORMAT_MY	"M/yy"	7/00
TIMEDATEFORMAT_Q	None	Q1
TIMEDATEFORMAT_MMMM	"MMMM"	January
TIMEDATEFORMAT_MMM	"MMM"	Jan
TIMEDATEFORMAT_M	"MMM"	J
TIMEDATEFORMAT_DDDD	"dddd"	Tuesday
TIMEDATEFORMAT_DDD	"ddd"	Tue
TIMEDATEFORMAT_D	"ddd"	T
TIMEDATEFORMAT_Y	"yy"	00
TIMEDATEFORMAT_MDY2000	"M/dd/yyyy"	12/07/2000
TIMEDATEFORMAT_DMY2000	"d/MM/yyyy"	7/12/2000

202 Axis Labels

TIMEDATEFORMAT_MY2000	"M/yyyy"	7/2000
TIMEDATEFORMAT_Y2000	"yyyy"	2000

In some cases, the TIMEDATEFORMAT_Q format for example, the **DateTime.ToString** class does not handle the desired conversion. In cases like this the date/time Format constant is trapped and undergoes additional processing to create the final label. That is why some of the date format strings are the same, even though the resulting labels are different.

TimeAxis Labels constructor

There is only one main constructor for **TimeAxisLabels** objects.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal baseaxis As TimeAxis _
)

[C#]
public TimeAxisLabels(
    TimeAxis baseaxis
);
```

baseaxis This is the time axis the axis labels are for.

Other axis label properties: font, rotation, numeric format, axis labels direction and numeric precision are automatically set. These properties can be explicitly set if you need to override the automatically calculated values.

SetAxisLabels method

```
[Visual Basic]
Overloads Public Sub SetAxisLabels( _
    ByVal font As Font, _
    ByVal rotation As Double, _
    ByVal labdir As Integer, _
    ByVal decimalpos As Integer, _
    ByVal labelends As Integer, _
    ByVal labcolor As Color _
)

[C#]
public void SetAxisLabels(
    Font font,
    double rotation,
    int labdir,
    int decimalpos,
    int labelends,
    Color labcolor
);
```

SetAxisLabelsFormat method

```

[Visual Basic]
Public Sub SetAxisLabelsFormat( _
    ByVal format As Integer _
)

[C#]
public void SetAxisLabelsFormat(
    int format
);

```

<i>font</i>	The font object used to display the axis label text.
<i>rotation</i>	The rotation, in degrees, of label text in the normal viewing plane.
<i>labdir</i>	The justification of the axis label (AXIS_MIN or AXIS_MAX) with respect to the tick mark endpoint.
<i>decimal</i>	Sets the number of digits to the right of the decimal point for numeric axis labels.
<i>labelends</i>	Ignored for time axis labels
<i>labcolor</i>	The color of the label text.
<i>format</i>	Sets the numeric format for the axis labels. Use one of the time format constants: TIMEDATEFORMAT_MSDDD, TIMEDATEFORMAT_MSDD, TIMEDATEFORMAT_MSD, TIMEDATEFORMAT_MS, TIMEDATEFORMAT_12HMSDD, TIMEDATEFORMAT_12HMSD, TIMEDATEFORMAT_12HMS, TIMEDATEFORMAT_12HM, TIMEDATEFORMAT_24HMSDD, TIMEDATEFORMAT_24HMSD, TIMEDATEFORMAT_24HMS, TIMEDATEFORMAT_24HM, TIMEDATEFORMAT_STANDARD, TIMEDATEFORMAT_MDY, TIMEDATEFORMAT_DMY, TIMEDATEFORMAT_MY, TIMEDATEFORMAT_Q, TIMEDATEFORMAT_MMMM, TIMEDATEFORMAT_MMM, TIMEDATEFORMAT_M, TIMEDATEFORMAT_DDDD, TIMEDATEFORMAT_DDD, TIMEDATEFORMAT_D, TIMEDATEFORMAT_Y, TIMEDATEFORMAT_MDY2000, TIMEDATEFORMAT_DMY2000, TIMEDATEFORMAT_MY2000, TIMEDATEFORMAT_Y2000.

Simple time axis labels example**[C#]**

```
// Define a Time coordinate system
ChartCalendar xMin = new ChartCalendar(1996, ChartObj.FEBRUARY, 5);
ChartCalendar xMax = new ChartCalendar(2002, ChartObj.JANUARY, 5);
double yMin = 0;
double yMax = 105;

TimeCoordinates simpleTimeScale;
simpleTimeScale = new TimeCoordinates(xMin, yMin, xMax, yMax);
// Create the time axis (x-axis is assumed)
TimeAxis xAxis = new TimeAxis(simpleTimeScale);
// Create the linear y-axis
LinearAxis yAxis =
new LinearAxis(simpleTimeScale, ChartObj.Y_AXIS);
TimeAxisLabels xAxisLabels = new TimeAxisLabels(xAxis);
NumericAxisLabels yAxisLabels = new NumericAxisLabels(yAxis);

// Create the ChartView object to place graph objects in.
ChartView chartVu = new ChartView();

// Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis);
chartVu.AddChartObject(yAxis);
chartVu.AddChartObject(xAxisLabels);
chartVu.AddChartObject(yAxisLabels);
```

[Visual Basic]

```
` Define a Time coordinate system
Dim xMin As ChartCalendar = New ChartCalendar(1996, ChartObj.FEBRUARY, 5)
Dim xMax As ChartCalendar = New ChartCalendar(2002, ChartObj.JANUARY, 5)
Dim yMin As Double = 0
Dim yMax As Double = 105

Dim simpleTimeScale As TimeCoordinates
simpleTimeScale = New TimeCoordinates(xMin, yMin, xMax, yMax)
' Create the time axis (x-axis is assumed)
```



```

Dim xAxis As TimeAxis = New TimeAxis(simpleTimeScale)
' Create the linear y-axis
Dim yAxis As LinearAxis = _
    New LinearAxis(simpleTimeScale, ChartObj.Y_AXIS)
Dim xAxisLabels As TimeAxisLabels = New TimeAxisLabels(xAxis)
Dim yAxisLabels As NumericAxisLabels = New NumericAxisLabels(yAxis)

' Create the ChartView object to place graph objects in.
Dim chartVu As ChartView = New ChartView()

' Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis)
chartVu.AddChartObject(yAxis)
chartVu.AddChartObject(xAxisLabels)
chartVu.AddChartObject(yAxisLabels)

```

Custom time axis labels example

[C#]

```

Font labelfont = new Font("Helvetica", 10, FontStyle.BOLD);

double xAxisLabelsRotation = 0.0;
int xAxisLabelsDir = ChartObj.AXIS_MIN;
int xAxisLabelsEnds = ChartObj.LABEL_ALL;
Color xAxisLabelsColor = Color.Black;
int xAxisNumericFormat = ChartObj.TIMEDATEFORMAT_MY;

xAxisLabels.SetAxisLabels( labelfont, xAxisLabelsRotation,
                           xAxisLabelsDir,
                           xAxisLabelsEnds, xAxisLabelsColor);
xAxisLabels.SetAxisLabelsFormat(xAxisNumericFormat);

```

[Visual Basic]

```

Dim labelfont As Font = New Font("Helvetica", 10, FontStyle.Bold)
Dim xAxisLabelsRotation As Double = 0.0
Dim xAxisLabelsDir As Integer = ChartObj.AXIS_MIN
Dim xAxisLabelsEnds As Integer = ChartObj.LABEL_ALL

```

```

Dim xAxisLabelsColor As Color = Color.Black
Dim xAxisNumericFormat As Integer = ChartObj.TIMEDATEFORMAT_MY

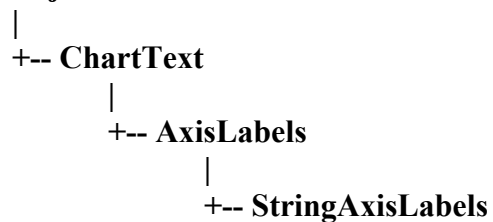
xAxisLabels.SetAxisLabels(labelfont, xAxisLabelsRotation, _
    xAxisLabelsDir, xAxisLabelsEnds, xAxisLabelsColor)
xAxisLabels.SetAxisLabelsFormat(xAxisNumericFormat)

```

String Axis Labels

Class StringAxisLabels

GraphObj



The **StringAxisLabels** class extends the **AxisLabels** class, adding extensive numeric. The class will label the major tick marks of a linear axis with user-defined strings.

StringAxisLabels constructor

There is only one main constructor for **StringAxisLabels** objects.

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal baseaxis As Axis _
)

[C#]
public StringAxisLabels(
    Axis baseaxis
);

```

baseaxis This is the axis the axis labels are for.

Other axis label properties: font, rotation and axis labels direction are automatically set. These properties can be explicitly set if you need to override the automatically calculated values.

SetAxisLabels method

```

[Visual Basic]
Overloads Public Sub SetAxisLabels( _
    ByVal font As Font, _
    ByVal rotation As Double, _
    ByVal labdir As Integer, _
    ByVal labelends As Integer, _

```

```

    ByVal labickcolor As Color, _
    ByVal tickstrings As String(), _
    ByVal numtickstrings As Integer _
)

```

```

[C#]
public void SetAxisLabels(
    Font font,
    double rotation,
    int labdir,
    int labelends,
    Color labcolor,
    string[] tickstrings,
    int numtickstrings
);

```

SetAxisLabelsStrings method

```

[Visual Basic]
Public Sub SetAxisLabelsStrings( _
    ByVal tickstrings As String(), _
    ByVal n As Integer _
)

```

```

[C#]
public void SetAxisLabelsStrings(
    string[] tstrings,
    int n
);

```

<i>font</i>	The font object used to display the axis label text.
<i>rotation</i>	The rotation, in degrees, of label text in the normal viewing plane.
<i>labdir</i>	The justification of the axis label (AXIS_MIN or AXIS_MAX) with respect to the tick mark endpoint.
<i>labelends</i>	Specifies whether there should be labels for the axis minimum (LABEL_MIN), maximum (LABEL_MAX) or tick mark starting point (LABEL_ORIGIN). The value of these constants can be OR'd together. The value of LABEL_MIN LABEL_MAX LABEL_ORIGIN is LABEL_ALL
<i>labcolor</i>	The color of the label text.
<i>tickstrings</i>	An array of strings, one for each major tick mark of the related axis. If you want to skip the first (or any) major tick mark, use an empty, "", string as the array element.

**StringLabel (extracted from the example program
Bargraphs.LandOfTheFry.InitializeVerticalChart)****[C#]**

```
String [] CountryNames =
    { "", "Brazil", "France",
      "Australia", "Spain", "United States", "England" };

// X-axis string labels
// Each string will label a major tick mark
StringAxisLabels xAxisLab1 = new StringAxisLabels(xAxis1);
xAxisLab1.SetAxisLabels(theFont, 0,
    ChartObj.AXIS_MIN, ChartObj.LABEL_ALL,
    Color.Black, CountryNames, 7);
xAxisLab1.SetColor(Color.Black);
chartVu.AddChartObject(xAxisLab1);
```

[VB]

```
Dim CountryNames As String() = { "", "Brazil", "France", "Australia", "Spain",
    "United States", "England" }

' X-axis string labels
' Each string will label a major tick mark
Dim xAxisLab1 As New StringAxisLabels(xAxis1)
xAxisLab1.SetAxisLabels(theFont, 0, ChartObj.AXIS_MIN, _
    ChartObj.LABEL_ALL, Color.Black, CountryNames, 7)
xAxisLab1.SetColor(Color.Black)
chartVu.AddChartObject(xAxisLab1)
```


9. Axis Grids

Grid

Axis grids are solid, dotted or dashed lines, aligned with the axis tick marks and which extend across the plot area volume of a graph. Axis grids are a separate class from the axis classes. An axis class, i.e. any class derived from **Axis**, can exist independent of a grid. Many graphs use axes that do not have grids. For example, the y-axis may have a grid, while the x-axis may not. The axis grid classes are not independent and they require valid references to two axes.

Linear, Logarithmic and Time Axis Grids

Class Grid

GraphObj

|
+-- **Grid**

The **Grid** class defines a grid for **LinearAxis**, **LogAxis**, and **TimeAxis** classes. A grid object needs a reference to two axes (x- and y-axis, x- and z-axis, or y- and z-axis). The grid aligns with the tick marks of one axis, and extends across the plot area volume using the minimum and maximum values of the other axis. For example, an x-axis grid has lines aligned with the tick marks of the x-axis and extending from the minimum y-value to the maximum y-value of the y-axis, parallel to the y-axis.

The default grid assumes that the grid is drawn in the XY_MAXZ_PLANE, the back plane of most 3D graphs. The default can be changed using the **SetGridAxesPlane** method.

Grid constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal axisref1 As Axis, _
    ByVal axisref2 As Axis, _
    ByVal gridaxistype As Integer, _
    ByVal gridtype As Integer _
)

[C#]
public Grid(
    Axis axisref1,
    Axis axisref2,
    int gridaxistype,
```

```
    int gridtype
);
```

<i>axisref1</i>	The first of two axes associated with the grid.
<i>axisref2</i>	The second of two axes associated with the grid.
<i>gridaxistype</i>	The grid is aligned with the tick marks of this axis. The grid is parallel to the other axis. Use one of the axis constants, X_AXIS, Y_AXIS or Z_AXIS.
<i>gridtype</i>	Specifies if the grid aligns with the major tick marks (GRID_MAJOR), the minor tick marks (GRID_MINOR) or the major and minor tick marks (GRID_ALL) of the reference axis.

Other grid properties are associated with the line properties used to draw the grid. The default values of the grid use a black dotted line of thickness 1.0. Change the default values using the **GraphObj** methods below.

SetColor method

```
[Visual Basic]
Overridable Public Sub SetColor( _
    ByVal rgbcolor As Color _
)

[C#]
public virtual void SetColor(
    Color rgbcolor
);
```

SetLineWidth method

```
[Visual Basic]
Overridable Public Sub SetLineWidth( _
    ByVal linewidth As Double _
)

[C#]
public virtual void SetLineWidth(
    double linewidth
);
```

SetLineStyle method

```
[Visual Basic]
Overridable Public Sub SetLineStyle( _
    ByVal linestyle As DashStyle _
)

[C#]
public virtual void SetLineStyle(
```



```
DashStyle linestyle
);
```

<i>rgbcolor</i>	Sets the primary line color for the chart object.
<i>linewidth</i>	Sets the line width, in device coordinates, for the chart object.
<i>linestyle</i>	Sets the line style for the chart object. Use one of the .Net Drawing2D.DashStyle enumerated constants: Dash, DashDot, DashDotDot, Dot or Solid.

Grid example – Extracted from the example program Bargraphs.SimpleBarChart

[C#]

```
// Auto-scale a coordinate system for data
pTransform1 = new TimeCoordinates();

.
.
.

// Define the x- or time, axis
TimeAxis xAxis = new TimeAxis(pTransform1);
chartVu.AddChartObject(xAxis);

// Define the y-axis
LinearAxis yAxis = new LinearAxis(pTransform1, ChartObj.Y_AXIS);
chartVu.AddChartObject(yAxis);

// Define the z-axis
LinearAxis zAxis = new LinearAxis(pTransform1, ChartObj.Z_AXIS);
zAxis.SetColor(Color.Black);
zAxis.AxisIntercept = xAxis.AxisMax;
zAxis.AxisIntercept2 = yAxis.AxisMin;
zAxis.AxisTickSpace = 0.25;
zAxis.AxisMinorTicksPerMajor = 1;
chartVu.AddChartObject(zAxis);

.
.
.
```

```

// Define the x-axis grid for the XY_MAXZ_PLANE
Grid xgrid = new Grid(xAxis, yAxis, ChartObj.X_AXIS, ChartObj.GRID_MAJOR);
xgrid.SetColor(Color.White);
xgrid.SetLineWidth(1);
xgrid.SetLineStyle(DashStyle.Solid);
chartVu.AddChartObject(xgrid);

// Define the y-axis grid for the XY_MAXZ_PLANE
Grid ygrid = new Grid(xAxis, yAxis, ChartObj.Y_AXIS, ChartObj.GRID_MAJOR);
ygrid.SetColor(Color.White);
ygrid.SetLineStyle(DashStyle.Solid);
ygrid.SetLineWidth(1);
chartVu.AddChartObject(ygrid);

// Define the z-axis grid for the YZ_MINX_PLANE
Grid zgrid1 = new Grid(xAxis, yAxis, zAxis, ChartObj.Z_AXIS, ChartObj.GRID_MAJOR);
zgrid1.SetColor(Color.White);
zgrid1.SetLineStyle(DashStyle.Solid);
zgrid1.SetLineWidth(1);
zgrid1.GridAxisPlane = ChartObj.YZ_MINX_PLANE;
chartVu.AddChartObject(zgrid1);

// Define the z-axis grid for the XZ_MINY_PLANE
Grid zgrid2 = new Grid(xAxis, zAxis, ChartObj.Z_AXIS, ChartObj.GRID_MAJOR);
zgrid2.SetColor(Color.White);
zgrid2.SetLineStyle(DashStyle.Solid);
zgrid2.SetLineWidth(1);
zgrid2.GridAxisPlane = ChartObj.XZ_MINY_PLANE;
chartVu.AddChartObject(zgrid2);

// Define the z-axis grid for the XZ_MINY_PLANE
Grid bottomxgrid = new Grid(xAxis, zAxis, ChartObj.X_AXIS, ChartObj.GRID_MAJOR);
bottomxgrid.SetColor(Color.White);
bottomxgrid.SetLineStyle(DashStyle.Solid);
bottomxgrid.SetLineWidth(1);
bottomxgrid.GridAxisPlane = ChartObj.XZ_MINY_PLANE;
chartVu.AddChartObject(bottomxgrid);

```

[Visual Basic]

```

` Auto-scale a coordinate system for data
Dim pTransform1 As New TimeCoordinates()
.

```

```

.
.

' Define the x- or time, axis
Dim xAxis As New TimeAxis(pTransform1)
xAxis.AxisTickDir3D = ChartObj.Y_AXIS
xAxis.AxisTickDir = ChartObj.AXIS_MIN
chartVu.AddChartObject(xAxis)

' Define the y-axis
Dim yAxis As New LinearAxis(pTransform1, ChartObj.Y_AXIS)
yAxis.AxisIntercept2 = 0.0
chartVu.AddChartObject(yAxis)

' Define the z-axis
Dim zAxis As New LinearAxis(pTransform1, ChartObj.Z_AXIS)
zAxis.SetColor(Color.Black)
zAxis.AxisIntercept = xAxis.AxisMax
zAxis.AxisIntercept2 = yAxis.AxisMin
zAxis.AxisTickSpace = 0.25
zAxis.AxisMinorTicksPerMajor = 1
chartVu.AddChartObject(zAxis)

' Define the x-axis time labels
Dim xAxisLab As New TimeAxisLabels(xAxis)
xAxisLab.SetAxisLabelsFormat(ChartObj.TIMEDATEFORMAT_Y2000)
xAxisLab.SetColor(Color.Black)
xAxisLab.TextRotation = 90
chartVu.AddChartObject(xAxisLab)

' Define the y-axis numeric labels
Dim yAxisLab As New NumericAxisLabels(yAxis)
yAxisLab.SetColor(Color.Black)
yAxisLab.SetAxisLabelsFormat(ChartObj.CURRENCYFORMAT)
chartVu.AddChartObject(yAxisLab)

' Define the z-axis string labels
Dim zlabels As String() = {"", "Cones", "Cylinders", "Bars"}
Dim zAxisLab As New StringAxisLabels(zAxis)
zAxisLab.SetColor(Color.Black)
zAxisLab.SetAxisLabels(theFont, 0, ChartObj.AXIS_MIN, ChartObj.LABEL_ALL,
Color.Black, zlabels, 4)
chartVu.AddChartObject(zAxisLab)

```

```

' Define the y-axis title
Dim yaxistitle As New AxisTitle(yAxis, theFont, "Millions $")
yaxistitle.SetColor(Color.Black)
chartVu.AddChartObject(yaxistitle)

' Define the z-axis title
Dim zaxistitle As New AxisTitle(zAxis, theFont, "Category")
zaxistitle.SetColor(Color.Black)
chartVu.AddChartObject(zaxistitle)

' Define the x-axis grid for the XY_MAXZ_PLANE
Dim xgrid As New Grid(xAxis, yAxis, ChartObj.X_AXIS, ChartObj.GRID_MAJOR)
xgrid.SetColor(Color.White)
xgrid.SetLineWidth(1)
xgrid.SetLineStyle(DashStyle.Solid)
chartVu.AddChartObject(xgrid)

' Define the y-axis grid for the XY_MAXZ_PLANE
Dim ygrid As New Grid(xAxis, yAxis, ChartObj.Y_AXIS, ChartObj.GRID_MAJOR)
ygrid.SetColor(Color.White)
ygrid.SetLineStyle(DashStyle.Solid)
ygrid.SetLineWidth(1)
chartVu.AddChartObject(ygrid)

' Define the z-axis grid for the YZ_MINX_PLANE
Dim zgrid1 As New Grid(xAxis, yAxis, zAxis, ChartObj.Z_AXIS, ChartObj.GRID_MAJOR)
zgrid1.SetColor(Color.White)
zgrid1.SetLineStyle(DashStyle.Solid)
zgrid1.SetLineWidth(1)
zgrid1.GridAxisPlane = ChartObj.YZ_MINX_PLANE
chartVu.AddChartObject(zgrid1)

' Define the z-axis grid for the XZ_MINY_PLANE
Dim zgrid2 As New Grid(xAxis, zAxis, ChartObj.Z_AXIS, ChartObj.GRID_MAJOR)
zgrid2.SetColor(Color.White)
zgrid2.SetLineStyle(DashStyle.Solid)
zgrid2.SetLineWidth(1)
zgrid2.GridAxisPlane = ChartObj.XZ_MINY_PLANE
chartVu.AddChartObject(zgrid2)

' Define the z-axis grid for the XZ_MINY_PLANE

```

```
Dim bottomxgrid As New Grid(xAxis, zAxis, ChartObj.X_AXIS, ChartObj.GRID_MAJOR)
bottomxgrid.SetColor(Color.White)
bottomxgrid.SetLineStyle(DashStyle.Solid)
bottomxgrid.SetLineWidth(1)
bottomxgrid.GridAxisPlane = ChartObj.XZ_MINI_PLANE
chartVu.AddChartObject(bottomxgrid)
```


10. Simple Plot Objects

SimplePlot

- SimpleBarPlot**
- SimpleLineMarkerPlot**
- SimpleLinePlot**
- SimpleRibbonPlot**
- SimplePipelinePlot**
- SimpleScatterPlot**

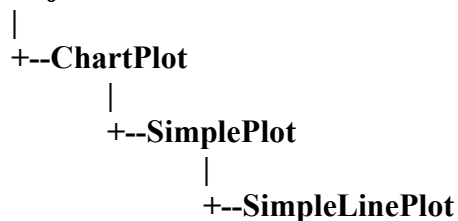
The **SimplePlot** class is an abstract class representing plot types that use data organized as a simple array of xy and/or z points. Simple plot types include: line plots, scatter plots, bar plots, ribbon plots, and line-marker plots. If the **SimplePlot** is attached to one of the 2½ D dataset types (**SimpleDataset** and **TimeSimpleDataset**), the plot will reside in a fixed z-axis plane. If the **SimplePlot** type is attached to one of the 3D dataset types (**SimpleDataset3D** and **TimeSimpleDataset3D**), it can display a plot that varies in the x-, y-, and z-dimension, point to point. When used in the simplest mode, simple plot objects use a single **ChartAttribute** object to control the plot objects color, line and fill styles. In terms of memory usage, this is the most efficient method. If memory is not an issue, it is also possible to assign every line segment, bar and scatter plot symbol a unique **ChartAttribute** object. Used in this mode, a single line plot can have unlimited number of multi-colored line segments. Another option labels each data point with its numeric y-value.

Example program segments presented in this documentation are not complete programs and contain uninitialized and/or undefined objects and variables. Do not attempt to copy them into your own program. Refer to the referenced example.

Simple Line Plots

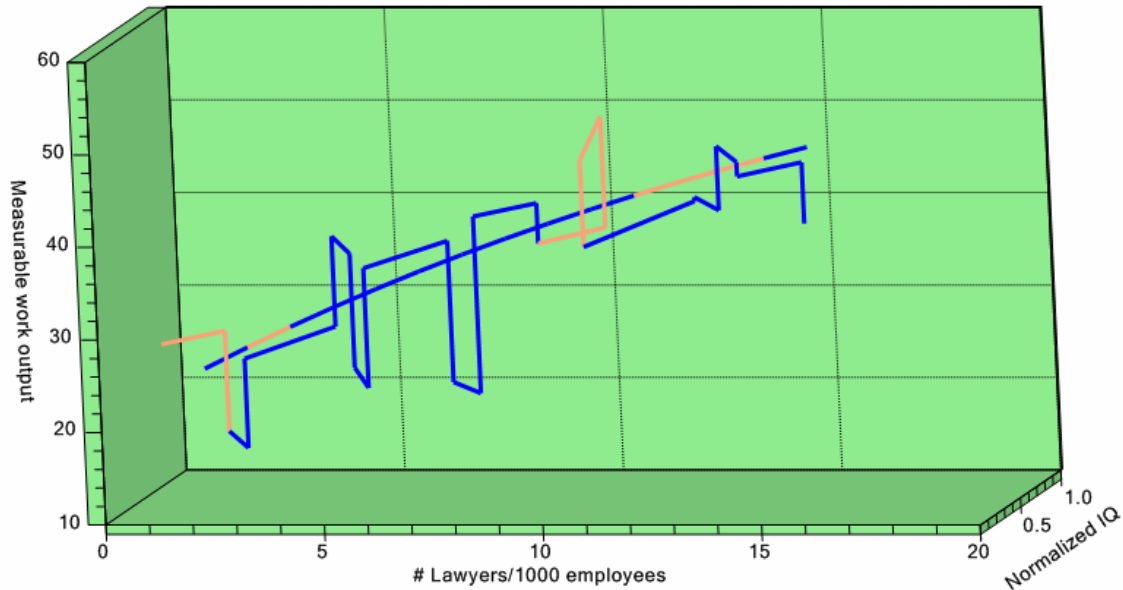
Class SimpleLinePlot

GraphObj



The **SimpleLinePlot** class is a subclass of the **SimplePlot** class and displays simple datasets in line plot format. Data points are connected using a straight line, or a step line. When used with a 2½ D dataset, the position of the line with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. Since a 2½ D simple line plot has

no depth, the attached datasets **ImplicitDepthValue** property is ignored. When a simple line plot is attached to a 3D dataset, the line is not restricted to a z-axis plane and can traverse 3D space unrestricted. In that case both the **ImplicitZValue** and **ImplicitDepthValue** are ignored.



A SimpleLinePlot attached to a SimpleDataset3D (SimpleLinePlots.LinePlot3D)

SimpleLinePlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As SimpleDataset, _
    ByVal attrib As ChartAttribute _
)

[C#]
public SimpleLinePlot(
    PhysicalCoordinates transform,
    SimpleDataset dataset,
    ChartAttribute attrib
);
```

transform The coordinate system for the new **SimpleLinePlot** object.

dataset The line plot represents the values in this dataset.

attrib Specifies the attributes (line color, thickness and style, fill color and fill mode) for the line plot.

A **ChartAttribute** object sets the objects global line color, line thickness, line style, fill color and fill mode. Change the **ChartAttribute** object using the objects **SetChartObjAttributes** method. There is also a group of methods that set individual simple plot properties: **SetColor**, **SetLineWidth**, and **SetLineStyle**. The line step style is using the **SetStepMode** method.

Individual line segments in a simple line plot object can have unique properties. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods.

Simple 2 1/2D line plot example (extracted from the example program ScatterPlots.ScatterPoints)

[C#]

```
TimeSimpleDataset Dataset2 = new TimeSimpleDataset("Second",x1,y2);

// Create a coordinate system for the data
TimeCoordinates pTransform1 = new TimeCoordinates( ChartObj.TIME_SCALE,
ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);

// Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, .15, .90, 0.8) ;

// Set rotational parameters
Point3D protate = new Point3D(10, 20, 0);
pTransform1.AbsRotateCoordinateSystem( protate);
pTransform1.TheTransform3D.ActualLightSource = new Point3D(0,0.0, 1);
// Set chart depth
chartVu.SetFractionalZViewportDepth(0.1);
.
.
.
// Create a simple line plot
ChartAttribute attrib2 = new ChartAttribute (Color.Green, 2,DashStyle.Solid);
SimpleLinePlot thePlot2 = new SimpleLinePlot(pTransform1, Dataset2, attrib2);
chartVu.AddChartObject(thePlot2);
```

[Visual Basic]

```
Dim Dataset1 As New TimeSimpleDataset("First", x1, y1)
```

222 *Simple Plot Objects*

```
Dataset1.ImplicitZValue = 0.5
Dim Dataset2 As New TimeSimpleDataset("Second", x1, y2)

' Create a coordinate system for the data
Dim pTransform1 As New TimeCoordinates(ChartObj.TIME_SCALE, ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

' Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 0.8)

' Set rotational parameters
Dim protate As New Point3D(10, 20, 0)
pTransform1.AbsRotateCoordinateSystem(protate)
pTransform1.TheTransform3D.ActualLightSource = New Point3D(0, 0.0, 1)

' Set chart depth
chartVu.SetFractionalZViewportDepth(0.1)
.
.
.

' Create a simple line plot
Dim attrib2 As New ChartAttribute(Color.Green, 2, DashStyle.Solid)
Dim thePlot2 As New SimpleLinePlot(pTransform1, Dataset2, attrib2)
chartVu.AddChartObject(thePlot2)
```

Simple 3D line plot example using segment colors (extracted from the example program SimpleLinePlots.LinePlot3D)

[C#]

```
int numPoints = 15;
double []x1 = new double[numPoints];
double []y1 = new double[numPoints];
double []y2 = new double[numPoints];
double []y3 = new double[numPoints];
double []z = new double[numPoints];
int i;

for (i=0; i < numPoints; i++)
{
    x1[i] = (double)(i+1) ;
    y1[i] = 20.0 + 50.0 * (1- Math.Exp(-x1[i]/20.0));
```

```

        y2[i] = y1[i] + (( 20+ 0.2* x1[i]) * (0.6 - ChartSupport.GetRandomDouble()));
        y3[i] = y1[i] + (( 20+ 0.4* x1[i]) * (0.4 - ChartSupport.GetRandomDouble()));
        z[i] = ChartSupport.GetRandomDouble();
    }

    // Create datasets

    SimpleDataset3D Dataset3 = new SimpleDataset3D("Third",x1,y3, z); // 3D

    // Scale coordinate system for data
    SimpleDataset [] datasetarray = {Dataset1, Dataset2, Dataset3};
    pTransform1 = new CartesianCoordinates( ChartObj.LINEAR_SCALE,
    ChartObj.LINEAR_SCALE);
    pTransform1.AutoScale(datasetarray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);

    // Position chart in graph area
    pTransform1.SetGraphBorderDiagonal(0.2, .2, .80, 0.725) ;

    // Set rotational parameters
    pTransform1.AbsRotateCoordinateSystem(new Point3D(10,15,0));
    .
    .
    .
    // Define third plot as a line plot using 3D data
    ChartAttribute attrib3 = new ChartAttribute (Color.Blue, 3,DashStyle.Solid);
    attrib3.SetSymbolSize(15);
    SimpleLinePlot thePlot3 = new SimpleLinePlot(pTransform1, Dataset3, attrib3);
    thePlot3.StepMode = ChartObj.STEP_END;
    thePlot3.SetSegmentAttributesMode(true);
    thePlot3.SetSegmentAttributes(8,segmentAttrib);
    thePlot3.SetSegmentAttributes(9,segmentAttrib);
    thePlot3.SetSegmentAttributes(0,segmentAttrib);
    chartVu.AddChartObject(thePlot3);

```

[Visual Basic]

```

Dim numPoints As Integer = 15
Dim x1(numPoints - 1) As Double
Dim y1(numPoints - 1) As Double
Dim y2(numPoints - 1) As Double
Dim y3(numPoints - 1) As Double
Dim z(numPoints - 1) As Double

```

224 Simple Plot Objects

```
Dim i As Integer

For i = 0 To numPoints - 1
    x1(i) = CDbl(i + 1)
    y1(i) = 20.0 + 50.0 * (1 - Math.Exp((-x1(i) / 20.0)))
    y2(i) = y1(i) + (20 + 0.2 * x1(i)) * (0.6 - ChartSupport.GetRandomDouble())
    y3(i) = y1(i) + (20 + 0.4 * x1(i)) * (0.4 - ChartSupport.GetRandomDouble())
    z(i) = ChartSupport.GetRandomDouble()
Next i

' Create datasets
Dim Dataset1 As New SimpleDataset("First", x1, y1) ' 2 1/2 D
Dim Dataset2 As New SimpleDataset3D("Second", x1, y2, z) ' 3D
Dim Dataset3 As New SimpleDataset3D("Third", x1, y3, z) ' 3D
' Scale coordinate system for data
Dim datasetarray As SimpleDataset() = {Dataset1, Dataset2, Dataset3}
Dim pTransform1 As _
    New CartesianCoordinates(ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(datasetarray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

' Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.2, 0.2, 0.8, 0.725)

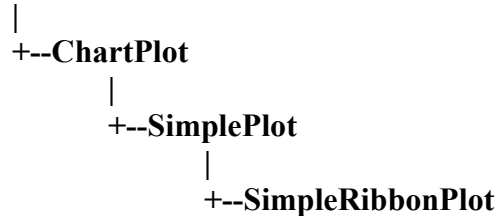
' Set rotational parameters
pTransform1.AbsRotateCoordinateSystem(New Point3D(10, 15, 0))
.
.
.

' Define second plot as a line plot using 2 1/2D data
Dim attrib2 As New ChartAttribute(Color.Blue, 3, DashStyle.Solid)
Dataset1.ImplicitZValue = 0.75
Dim thePlot2 As New SimpleLinePlot(pTransform1, Dataset1, attrib2)
thePlot2.SetSegmentAttributesMode(True)
thePlot2.SetSegmentAttributes(1, segmentAttrib)
thePlot2.SetSegmentAttributes(10, segmentAttrib)
thePlot2.SetSegmentAttributes(11, segmentAttrib)
thePlot2.SetSegmentAttributes(12, segmentAttrib)
chartVu.AddChartObject(thePlot2)
```

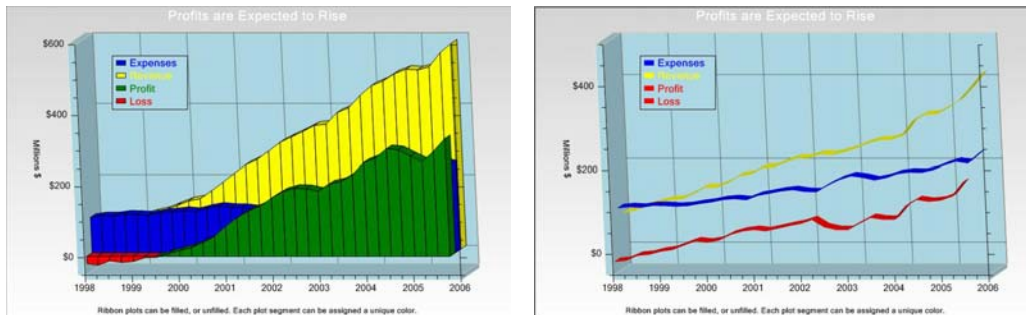
Simple Ribbon Plots

Class SimpleRibbonPlot

GraphObj



The **SimpleRibbonPlot** class is a subclass of the **SimplePlot** class and displays simple datasets in ribbon plot format. The ribbon plot is similar to a line plot, except that it has the added dimension of depth. Data points are connected using a straight ribbon, or a step ribbon. The position of the ribbon with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The depth of the ribbon is specified using the attached datasets **ImplicitDepthValue** property.



Examples of filled, and unfilled, ribbon plots (SimpleLinePlots.LineFill)

SimpleRibbonPlot constructor

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As SimpleDataset, _
    ByVal attrib As ChartAttribute _
)

[C#]
public SimpleRibbonPlot(
    PhysicalCoordinates transform,
    SimpleDataset dataset,
    ChartAttribute attrib
);
  
```

transform

The coordinate system for the new **SimpleRibbonPlot** object.

<i>dataset</i>	The ribbon plot represents the values in this dataset.
<i>attrib</i>	Specifies the attributes (line color, thickness and style, fill color and fill mode) for the ribbon plot.

A **ChartAttribute** object sets the objects global line color, line thickness, line style, fill color and fill mode. Change the **ChartAttribute** object using the objects **SetChartObjAttributes** method. There is also a group of methods that set individual simple plot properties: **SetColor**, **SetLineWidth**, and **SetLineStyle**. The line step style is using the **SetStepMode** method.

Individual line segments in a simple line plot object can have unique properties. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods.

Simple 2 1/2D ribbon plot example (extracted from the example program **SimpleLinePlots.LineFill**)

[C#]

```
// Simulate data and create datasets

// Initialize datasets to 0 size to test AddTimeDataPoint method below
TimeSimpleDataset Dataset1 = new TimeSimpleDataset("Sales",0);
Dataset1.ImplicitZValue = 0.0;
Dataset1.ImplicitDepthValue = 0.3;
.
.
.
TimeSimpleDataset[] DatasetArray = {Dataset1, Dataset2, Dataset3 };

// Create a coordinate system scaled for the data
pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(DatasetArray, ChartObj.AUTOAXES_FAR ,
ChartObj.AUTOAXES_FAR);

// Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, .1, .92, 0.75) ;

// Set rotational parameters
pTransform1.AbsRotateCoordinateSystem(new Point3D(10,15,0));
// Set chart 3D depth
```

```

chartVu.SetFractionalZViewportDepth(0.2);

.
.
.

// Define plot as a simple ribbon plot
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid);
Dataset1.SortByX(true);
thePlot1 = new SimpleRibbonPlot(pTransform1, Dataset1, attrib1);
thePlot1.SetLineStyle(DashStyle.DashDot);
chartVu.AddChartObject(thePlot1);

```

[Visual Basic]

```

' Simulate data and create datasets
' Initialize datasets to 0 size to test AddTimeDataPoint method below
Dim Dataset1 As New TimeSimpleDataset("Sales", 0)
Dataset1.ImplicitZValue = 0.0
Dataset1.ImplicitDepthValue = 0.3
.
.
.
Dim DatasetArray As TimeSimpleDataset() = {Dataset1, Dataset2, Dataset3}

' Create a coordinate system scaled for the data
pTransform1 = New TimeCoordinates()
pTransform1.AutoScale(DatasetArray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

' Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, 0.1, 0.92, 0.75)

' Set rotational parameters
pTransform1.AbsRotateCoordinateSystem(New Point3D(10, 15, 0))
' Set chart 3D depth
chartVu.SetFractionalZViewportDepth(0.2)
.
.
.
' Define plot as a simple ribbon plot
Dim attrib1 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid)
Dataset1.SortByX(True)
thePlot1 = New SimpleRibbonPlot(pTransform1, Dataset1, attrib1)

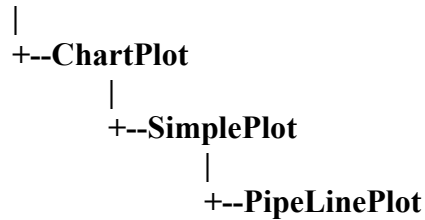
```

```
thePlot1.SetLineStyle(DashStyle.DashDot)
chartVu.AddChartObject(thePlot1)
```

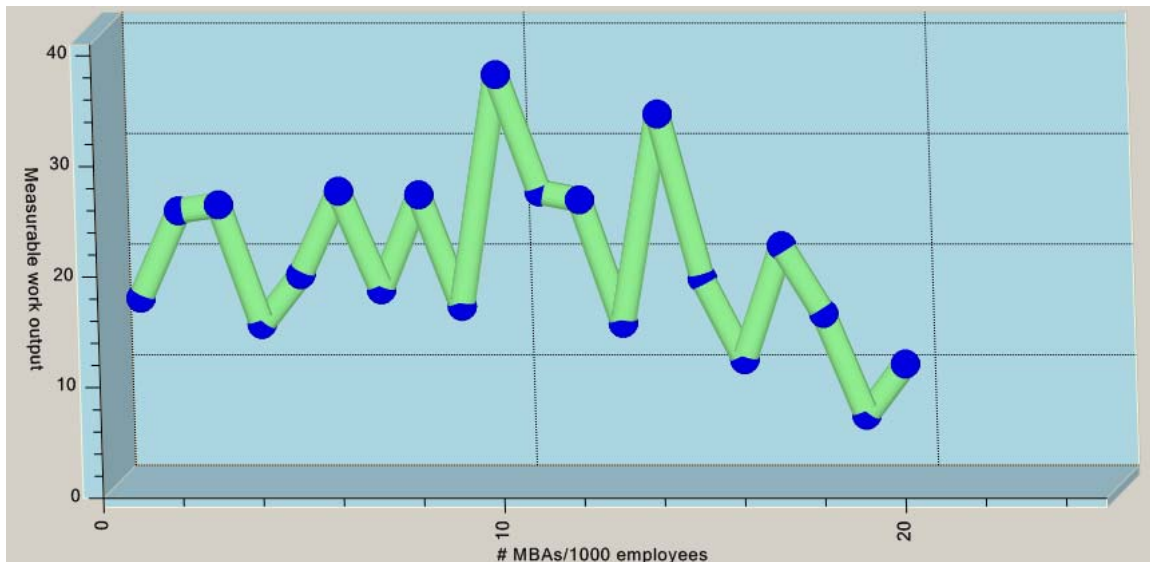
Simple Pipeline Plots

Class SimpleLinePlot

GraphObj



The **PipeLinePlot** class is a subclass of the **SimplePlot** class and displays simple datasets in pipeline plot format. The pipeline plot is similar to a line plot, except that the line is replaced by a 3D cylinder object with the added dimension of depth. Data points are connected using the cylinder. A spherical object connects adjacent pipes in the pipeline plot.



Pipeline plot (SimpleLinePlots.PipelineAndScatterPlot)

PipeLinePlot contructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As SimpleDataset, _
    ByVal attrib As ChartAttribute, _
```



```

        ByVal symbolattrib As ChartAttribute _
    )
    [C#]
    public PipeLinePlot(
        PhysicalCoordinates transform,
        SimpleDataset dataset,
        ChartAttribute attrib,
        ChartAttribute symbolattrib
    );

```

<i>transform</i>	The coordinate system for the new PipeLinePlot object.
<i>dataset</i>	The ribbon plot represents the values in this dataset.
<i>attrib</i>	Specifies the attributes (color) of the cylinder part of the pipeline plot.
<i>symbolattrib</i>	Specifies the attributes (color) of the connecting sphere symbol of the pipeline plot.

The radius of the cylinder is set using the PipeLinePlot's **BarWidth** property. The radius of the cylinder is set relative to the x-dimension, regardless of the orientation of the pipeline cylinders. This means if the x-dimension uses a time scale, you must set the radius of the cylinder in time coordinates, which is in milliseconds. This is demonstrated in the FinancialExamples.OHLCPlot example program/

A **ChartAttribute** object sets the color. Change the **ChartAttribute** object using the objects **SetChartObjAttributes** method. There is also a group of methods that set individual simple plot properties: **SetColor**.

Individual pipeline segments in a pipeline plot object can have unique properties. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods.

Simple 2 1/2D pipeline plot example (extracted from the example program SimpleLinePlots.PipeLineAndScatterPlot)

```
[C#]
```

```

SimpleDataset Dataset1 = new SimpleDataset("First",x1,y1);
SimpleDataset Dataset2 = new SimpleDataset("Second",x1,y2, 0.1, 0.2);
SimpleDataset Dataset3 = new SimpleDataset("Third",x1,y3, 0.9, 0.1);

// Define coordinate system based on data

```

230 *Simple Plot Objects*

```
CartesianCoordinates pTransform1 = new CartesianCoordinates(
ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);

pTransform1.AutoScale(Dataset3, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
// Add a bit to the auto-calculated y-scale
pTransform1.ScaleStopY += 1.0;

// Set rotational parameters
pTransform1.AbsRotateCoordinateSystem(new Point3D(10,10,0));
// Set chart depth
chartVu.SetFractionalZViewportDepth(0.2);

.
.
.
// Define second plot as a pipeline plot
ChartAttribute attrib2 = new ChartAttribute (Color.LightGreen, 3,DashStyle.Solid,
Color.LightGreen);
ChartAttribute attrib21 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid,
Color.Blue);
PipeLinePlot thePlot2 = new PipeLinePlot(pTransform1, Dataset2, attrib2,
attrib21);
thePlot2.BarWidth = 0.6;
chartVu.AddChartObject(thePlot2);
```

[VB]

```
' Create datasets
Dim Dataset1 As New SimpleDataset("First", x1, y1)
Dim Dataset2 As New SimpleDataset("Second", x1, y2, 0.1, 0.2)
Dim Dataset3 As New SimpleDataset("Third", x1, y3, 0.9, 0.1)

' Define coordinate system based on data
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _
ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset3, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
' Add a bit to the auto-calculated y-scale
pTransform1.ScaleStopY += 1.0

' Set rotational parameters
pTransform1.AbsRotateCoordinateSystem(New Point3D(10, 10, 0))
' Set chart depth
chartVu.SetFractionalZViewportDepth(0.2)

.
```

```

.
.

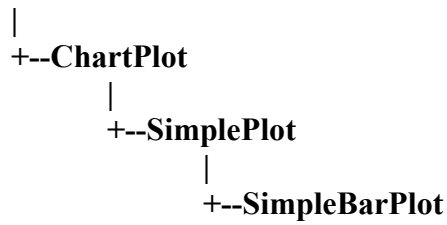
' Define second plot as a pipeline plot
Dim attrib2 As New ChartAttribute(Color.LightGreen, 3, DashStyle.Solid, _
    Color.LightGreen)
Dim attrib21 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid, Color.Blue)
Dim thePlot2 As New PipeLinePlot(pTransform1, Dataset2, attrib2, attrib21)
thePlot2.BarWidth = 0.6
chartVu.AddChartObject(thePlot2)

```

Simple Bar Plots

Class SimpleBarPlot

GraphObj



The **SimpleBarPlot** class is a subclass of the **SimplePlot** class and displays data in a 3D bar format. Individual bars, the maximum value of which corresponds to the y-values of the dataset, display justified with respect to the x-values. When used with a 2½ D dataset, the position of the bar with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The bar plot depth is set using the attached datasets **ImplicitDepthValue** property. When a simple bar plot is attached to a 3D dataset, the bar is not restricted to a z-axis plane and can traverse 3D space unrestricted. In that case the **ImplicitZValue** is ignored. The individual elements of the bar plot are normally 3D bars, but you can also set the bar plots **BarShapeOption** property to a cone (CONE3D) or cylinder (CYLINDER3D).



A bar plot that uses bars, cylinders and bars (*Bargraphs.SimpleBars*)

SimpleBarPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As SimpleDataset, _
    ByVal barwidth As Double, _
    ByVal barbase As Double, _
    ByVal attrib As ChartAttribute, _
    ByVal barjust As Integer _
)

[ C# ]
public SimpleBarPlot(
    PhysicalCoordinates transform,
    SimpleDataset dataset,
    double barwidth,
    double barbase,
    ChartAttribute attrib,
    int barjust
);
```

<i>transform</i>	The coordinate system for the new SimpleBarPlot object.
<i>dataset</i>	The bar plot represents the values in this dataset.
<i>barwidth</i>	The width of the bars in physical coordinates.
<i>barbase</i>	The base value for bars in physical coordinates.
<i>attrib</i>	Specifies the attributes (line color and fill color) of the bars.

barjust Specifies the justification with respect to the independent data value. Use one of the justification constants: JUSTIFY_MIN, JUSTIFY_CENTER, JUSTIFY_MAX.

A **ChartAttribute** object sets the outline line color, fill color and fill mode. Change the **ChartAttribute** object using the objects **SetChartObjAttributes** method. The simple bar plot **SetColor** method can be used to change the bar color.

Individual bars in a simple bar plot object can have unique properties. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods.

Simple bar plot example (extracted from the example program **Bargraphs.SimpleBars**)

[C#]

```
double bardepth = 0.1;
double z1 = 0.7, z2 = 0.45, z3 = 0.2;
TimeSimpleDataset Dataset1 = new TimeSimpleDataset("Sales #1",x1,y1, z1,
bardepth);
TimeSimpleDataset Dataset2 = new TimeSimpleDataset("Sales #2",x1,y2, z2,
bardepth);
TimeSimpleDataset Dataset3 = new TimeSimpleDataset("Sales #3",x1,y3, z3,
bardepth);

// An array of the datasets
TimeSimpleDataset [] datasetarray = {Dataset1, Dataset2, Dataset3};

// Auto-scale a coordinate system for data
pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(datasetarray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
.
.
.

// Define plot #1
ChartAttribute attrib1 = new ChartAttribute (Color.Green, 0,DashStyle.Solid,
Color.Green);
SimpleBarPlot thePlot1 = new SimpleBarPlot(pTransform1, Dataset1,
    ChartCalendar.GetCalendarWidthValue(ChartObj.MONTH,10), 0.0,
    attrib1, ChartObj.JUSTIFY_CENTER);
chartVu.AddChartObject(thePlot1);
```

234 *Simple Plot Objects*

```
// Define plot #2
ChartAttribute attrib2 = new ChartAttribute (Color.Crimson, 0,DashStyle.Solid,
Color.Crimson);
SimpleBarPlot thePlot2 = new SimpleBarPlot(pTransform1, Dataset2,
    ChartCalendar.GetCalendarWidthValue(ChartObj.MONTH,10), 0.0,
    attrib2, ChartObj.JUSTIFY_CENTER);
thePlot2.BarShapeOption = ChartObj.CYLINDER3D;
chartVu.AddChartObject(thePlot2);

// Define plot #3
ChartAttribute attrib3 = new ChartAttribute (Color.Black, 0,DashStyle.Solid,
Color.DodgerBlue);
SimpleBarPlot thePlot3 = new SimpleBarPlot(pTransform1, Dataset3,
    ChartCalendar.GetCalendarWidthValue(ChartObj.MONTH,10),
    0.0, attrib3, ChartObj.JUSTIFY_CENTER);
thePlot3.BarShapeOption = ChartObj.CONE3D;
chartVu.AddChartObject(thePlot3);
```

[Visual Basic]

```
' Create datasets with different implicit z-values, but the same depth
Dim bardepth As Double = 0.1
Dim z1 As Double = 0.7
Dim z2 As Double = 0.45
Dim z3 As Double = 0.2
Dim Dataset1 As New TimeSimpleDataset("Sales #1", x1, y1, z1, bardepth)
Dim Dataset2 As New TimeSimpleDataset("Sales #2", x1, y2, z2, bardepth)
Dim Dataset3 As New TimeSimpleDataset("Sales #3", x1, y3, z3, bardepth)

' An array of the datasets
Dim datasetarray As TimeSimpleDataset() = {Dataset1, Dataset2, Dataset3}

' Auto-scale a coordinate system for data
Dim pTransform1 As New TimeCoordinates()
pTransform1.AutoScale(datasetarray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
' force starting y-coordinate scale value of 0
pTransform1.SetScaleStartY(0)
' force starting z-coordinate calendar value
pTransform1.SetTimeScaleStart(New ChartCalendar(1997, ChartObj.JULY, 1))

' pTransform1.InvertScaleX();
```

```

' Set chart rotation
' Make the z-coordinate viewport depth to be 0.6 of the width of the viewport
chartVu.SetFractionalZViewportDepth(0.6)
.
.
.

' Define plot #1
Dim attrib1 As New ChartAttribute(Color.Green, 0, DashStyle.Solid, Color.Green)
Dim thePlot1 As New SimpleBarPlot(pTransform1, Dataset1, _
    ChartCalendar.GetCalendarWidthValue(ChartObj.MONTH, 10), _
    0.0, attrib1, ChartObj.JUSTIFY_CENTER)
chartVu.AddChartObject(thePlot1)

' Define plot #2
Dim attrib2 As New ChartAttribute(Color.Crimson, 0, _
    DashStyle.Solid, Color.Crimson)
Dim thePlot2 As New SimpleBarPlot(pTransform1, Dataset2, _
    ChartCalendar.GetCalendarWidthValue(ChartObj.MONTH, 10), _
    0.0, attrib2, ChartObj.JUSTIFY_CENTER)
thePlot2.BarShapeOption = ChartObj.CYLINDER3D
chartVu.AddChartObject(thePlot2)

' Define plot #3
Dim attrib3 As New ChartAttribute(Color.Black, 0, DashStyle.Solid, _
    Color.DodgerBlue)
Dim thePlot3 As New SimpleBarPlot(pTransform1, Dataset3, _
    ChartCalendar.GetCalendarWidthValue(ChartObj.MONTH, 10), _
    0.0, attrib3, ChartObj.JUSTIFY_CENTER)
thePlot3.BarShapeOption = ChartObj.CONE3D
chartVu.AddChartObject(thePlot3)

```

* Note how the **ChartCalendar.GetCalendarWidthValue** method calculates the width of the bars as a function of time, in this case a width of 8 months.

Simple bar plot example that displays numeric data values (extracted from the example program `Bargraphs.SimpleBars`)

[C#]

```

.
.

```

```

.
ChartAttribute attrib1 =
    new ChartAttribute (Color.Green, 0,DashStyle.Solid, Color.Green);
attrib1.SetFillFlag(true);
SimpleBarPlot thePlot1 = new SimpleBarPlot(pTransform1, Dataset1,
    ChartCalendar.GetCalendarWidthValue(ChartObj.MONTH,8), 0.0,
    attrib1, ChartObj.JUSTIFY_CENTER);
NumericLabel bardatavalue = thePlot1.GetPlotLabelTemplate();
bardatavalue.SetTextFont(theFont);
bardatavalue.SetNumericFormat(ChartObj.CURRENCYFORMAT);
bardatavalue.SetDecimalPos(0);
bardatavalue.SetColor(Color.White);
thePlot1.SetPlotLabelTemplate(bardatavalue);
thePlot1.SetShowDatapointValue(true);
chartVu.AddChartObject(thePlot1);

```

[Visual Basic]

```

.
.
.
Dim attrib1 As New ChartAttribute(Color.Green, 0, DashStyle.Solid, Color.Green)
attrib1.SetFillFlag(True)
Dim thePlot1 As New SimpleBarPlot(pTransform1, Dataset1, _
    ChartCalendar.GetCalendarWidthValue(ChartObj.MONTH, 8), _
    0.0, attrib1, ChartObj.JUSTIFY_CENTER)
Dim bardatavalue As NumericLabel = thePlot1.GetPlotLabelTemplate()
bardatavalue.SetTextFont(theFont)
bardatavalue.SetNumericFormat(ChartObj.CURRENCYFORMAT)
bardatavalue.SetDecimalPos(0)
bardatavalue.SetColor(Color.White)
thePlot1.SetPlotLabelTemplate(bardatavalue)
thePlot1.SetShowDatapointValue(True)
chartVu.AddChartObject(thePlot1)

```

Simple Scatter Plots

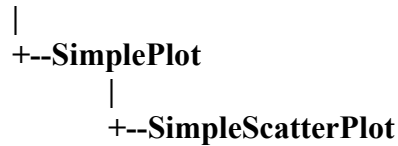
Class SimpleScatterPlot

GraphObj

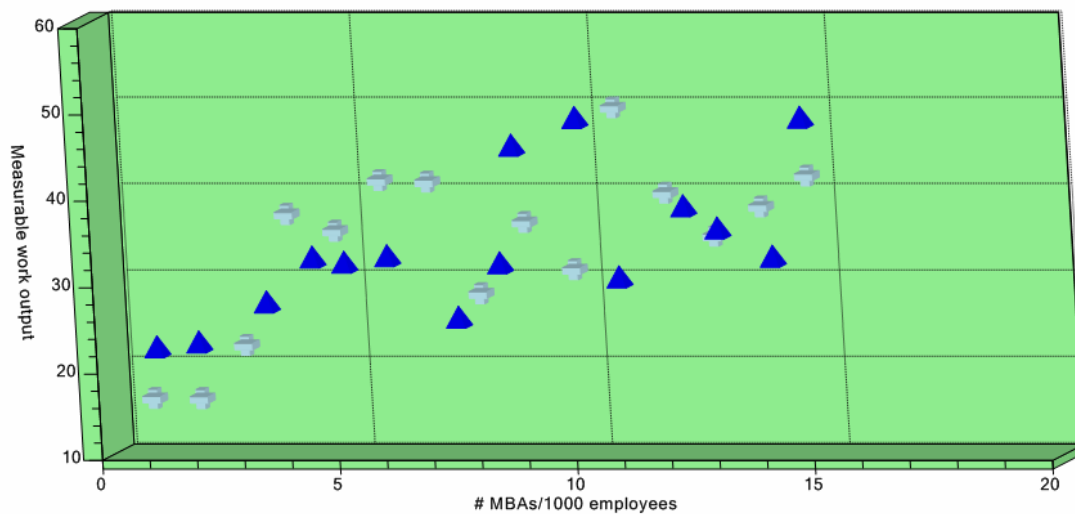
```

|
+--ChartPlot

```

The **SimpleScatterPlot** class is a subclass of the **SimplePlot** class and displays simple datasets in scatter plot format where each data point is a symbol. The symbols can take the form of a flat 2D symbol, or a 3D symbol. When used with a 2½ D dataset, the position of the symbol with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property is not used. When a simple scatter plot is attached to a 3D dataset, the scatter plot symbol is not restricted to a z-axis plane and can traverse 3D space unrestricted.



Simple scatter plots (ScatterPlots.SimpleScatter)

SimpleScatterPlot constructor

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As SimpleDataset, _
    ByVal symtype As Integer, _
    ByVal attrib As ChartAttribute _
)

[C#]
public SimpleScatterPlot(
    PhysicalCoordinates transform,
    SimpleDataset dataset,
    int symtype,
    ChartAttribute attrib
);

```

<i>transform</i>	The coordinate system for the new SimpleScatterPlot object.
<i>dataset</i>	The scatter plot represents the values in this dataset.
<i>symtype</i>	The symbol used in the scatter plot. Use one of the scatter plot symbol constants. 2D Symbols: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE. 3D Symbols: HBAR3D, VBAR3D, CUBE3D, PLUS3D, DIAMOND3D, PYRAMID3D, HOURGLASS3D, ARROW3D, CYLINDER3D, SPHERE3D
<i>attrib</i>	Specifies the attributes (size, line and fill color) for the scatter plot.

A **ChartAttribute** object sets the objects global outline and fill attributes. Change the simple plot objects **ChartAttribute** object using the objects **SetChartObjAttributes** method. For a simple color change of the scatter plot symbol, use the scatter plot objects **SetColor** method.

Individual scatter plot symbols in a scatter plot object can have unique properties. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods.

Simple scatter plot example (extracted from the example program **ScatterPlots.SimplePoints**)

[C#]

```
// Define dataset
TimeSimpleDataset Dataset1 = new TimeSimpleDataset("First",x1,y1);
Dataset1.ImplicitZValue = 0.5;
TimeSimpleDataset Dataset2 = new TimeSimpleDataset("Second",x1,y2);

// Create a coordinate system for the data
TimeCoordinates pTransform1 = new TimeCoordinates( ChartObj.TIME_SCALE,
ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);

// Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, .15, .90, 0.8) ;

// Set rotational parameters
Point3D protate = new Point3D(10, 20, 0);
```

```

pTransform1.AbsRotateCoordinateSystem( protate);
pTransform1.TheTransform3D.ActualLightSource = new Point3D(0,0.0, 1);
// Set chart depth
chartVu.SetFractionalZViewportDepth(0.1);
.
.
.
.
// Define a simple scatter plot
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid,
Color.Blue);
attrib1.SetSymbolSize(10);
SimpleScatterPlot thePlot1 = new SimpleScatterPlot(pTransform1, Dataset1,
ChartObj.CUBE3D, attrib1);
chartVu.AddChartObject(thePlot1);

```

[Visual Basic]

```

' Define dataset
Dim Dataset1 As New TimeSimpleDataset("First", x1, y1)
Dataset1.ImplicitZValue = 0.5
Dim Dataset2 As New TimeSimpleDataset("Second", x1, y2)

' Create a coordinate system for the data
Dim pTransform1 As New TimeCoordinates(ChartObj.TIME_SCALE, ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

' Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 0.8)

' Set rotational parameters
Dim protate As New Point3D(10, 20, 0)
pTransform1.AbsRotateCoordinateSystem(protate)
pTransform1.TheTransform3D.ActualLightSource = New Point3D(0, 0.0, 1)
' Set chart depth
chartVu.SetFractionalZViewportDepth(0.1)
.
.
.
' Define a simple scatter plot
Dim attrib1 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid, Color.Blue)
attrib1.SetSymbolSize(10)
Dim thePlot1 As New SimpleScatterPlot(pTransform1, Dataset1, _

```

```

        ChartObj.CUBE3D, attrib1)
chartVu.AddChartObject(thePlot1)

```

Simple scatter plot example that uses `SetSegmentAttributesMode` to change the size and color of individual scatter plot symbols in the plot (extracted from the example program `ScatterPlots.ScatterPoints`)

[C#]

```

.
.
.
// Define a simple scatter plot
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid,
Color.Blue);
attrib1.SetSymbolSize(10);
SimpleScatterPlot thePlot1 = new SimpleScatterPlot(pTransform1, Dataset1,
ChartObj.CUBE3D, attrib1);
// Change the attributes of individual scatter plot points
thePlot1.SetSegmentAttributesMode(true);
ChartAttribute segmentAttrib = new ChartAttribute (Color.Red, 1,DashStyle.Solid,
Color.Red);
segmentAttrib.SetSymbolSize(20);
thePlot1.SetSegmentAttributes(8,segmentAttrib);
thePlot1.SetSegmentAttributes(9,segmentAttrib);
thePlot1.SetSegmentAttributes(10,segmentAttrib);
thePlot1.SetSegmentAttributes(11,segmentAttrib);
chartVu.AddChartObject(thePlot1);

```

[Visual Basic]

```

.
.
.
' Define a simple scatter plot
Dim attrib1 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid, Color.Blue)
attrib1.SetSymbolSize(10)
Dim thePlot1 As New SimpleScatterPlot(pTransform1, Dataset1, _
    ChartObj.CUBE3D, attrib1)

' Change the attributes of individual scatter plot points
thePlot1.SetSegmentAttributesMode(True)
Dim segmentAttrib As New ChartAttribute(Color.Red, 1, DashStyle.Solid, Color.Red)
segmentAttrib.SetSymbolSize(20)

```

```

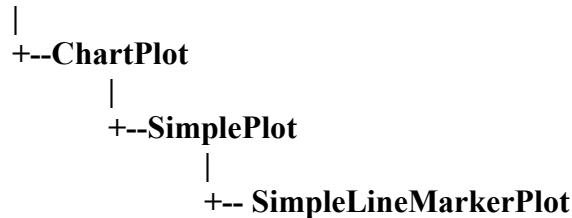
thePlot1.SetSegmentAttributes(8, segmentAttrib)
thePlot1.SetSegmentAttributes(9, segmentAttrib)
thePlot1.SetSegmentAttributes(10, segmentAttrib)
thePlot1.SetSegmentAttributes(11, segmentAttrib)
chartVu.AddChartObject(thePlot1)

```

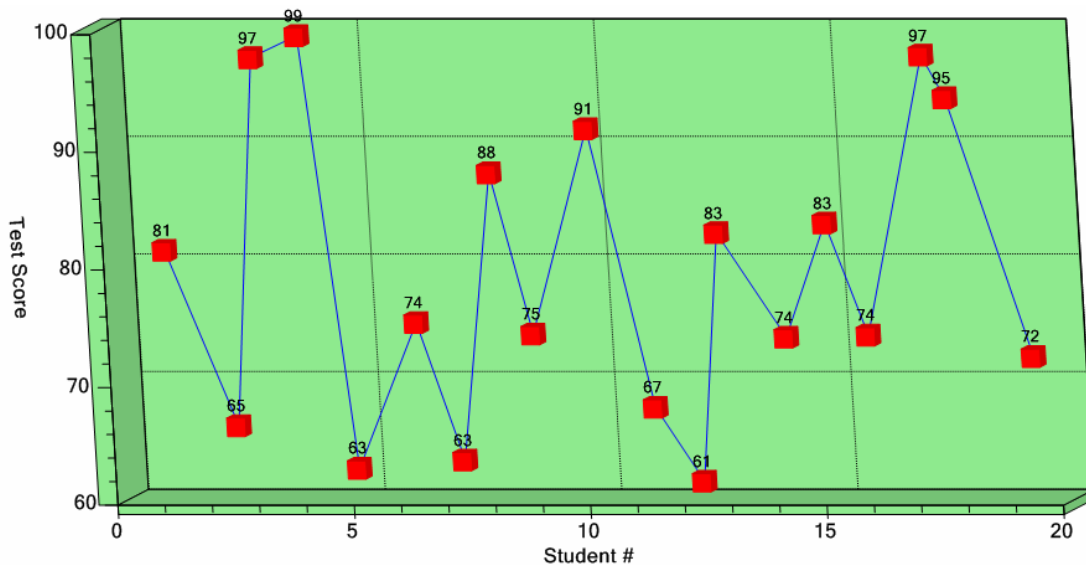
Simple Line Marker Plots

Class SimpleLineMarkerPlot

GraphObj



The **SimpleLineMarkerPlot** class is a subclass of the **SimplePlot** class and displays simple datasets in a line plot format where scatter plot symbols highlight individual data points. The symbols can take the form of a flat 2D symbol, or a 3D symbol. When used with a 2½ D dataset, the position of the symbol with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property is not used. When a simple scatter plot is attached to a 3D dataset, the scatter plot symbol is not restricted to a z-axis plane and can traverse 3D space unrestricted.



Simple line marker plot (ScatterPlots.LabeledDatapoints)

SimpleLineMarkerPlot constructor

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal attrs As ChartAttribute() _
)

[C#]
public StackedLinePlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    ChartAttribute[] attrs
);

```

<i>transform</i>	The coordinate system for the new SimpleLineMarkerPlot object.
<i>dataset</i>	The line marker plot represents the values in this dataset.
<i>symtype</i>	The symbol used in the scatter plot. Use one of the scatter plot symbol constants. 2D Symbols: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE. 3D Symbols: HBAR3D, VBAR3D, CUBE3D, PLUS3D, DIAMOND3D, PYRAMID3D, HOURGLASS3D, ARROW3D, CYLINDER3D, SPHERE3D
<i>.lineattrib</i>	Specifies the attributes (line color and line style) for the line part of the line marker plot.
<i>symbolattrib</i>	Specifies the attributes (line and fill color) for the symbol part of the line marker plot.
<i>nsymbolstart</i>	Specifies the starting index for symbols in the line marker plot.
<i>nsymbolskip</i>	Specifies the skip factor for placing symbols in the line marker plot.

A **ChartAttribute** object sets the objects global line color, line width and line style attributes. Change the **ChartAttribute** object using the objects **SetChartObjAttributes** method. Use the objects **SetSymbolAttributes** to change the attributes of the marker symbol.

Individual line segments in a line marker plot object can have unique properties. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods. If this option is used, the line and fill properties of the lines and the marker symbols will be the same.

Simple line marker plot example (extracted from the example program ScatterPlots.LabeledDatapoints)

[C#]

```
// Create a 3D dataset
SimpleDataset3D Dataset1 = new SimpleDataset3D("First",x1,y1, z1);
// Create a coordinate system to match data
CartesianCoordinates pTransform1 = new CartesianCoordinates(
ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
// Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.125, .15, .90, 0.8) ;
// Set rotate parameters
Point3D protate = new Point3D(10, 20, 0);
pTransform1.AbsRotateCoordinateSystem( protate);
// Set chart depth
chartVu.SetFractionalZViewportDepth(0.1);

.
.
.

// Define a simple line marker plot
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid);
ChartAttribute attrib2 = new ChartAttribute (Color.Red, 1,DashStyle.Solid);
attrib2.SetFillColor (Color.Red);
attrib2.SetFillFlag (true);
attrib2.SetSymbolSize(15);
SimpleLineMarkerPlot thePlot1 =
new SimpleLineMarkerPlot(pTransform1, Dataset1, ChartObj.CUBE3D, attrib1,attrib2,
0, 1);

chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
` Create a 3D dataset
Dim Dataset1 As New SimpleDataset3D("First", x1, y1, z1)
' Create a coordinate system to match data
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE,
ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
```

```

' Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.125, 0.15, 0.9, 0.8)

' Set rotate parameters
Dim protate As New Point3D(10, 20, 0)
pTransform1.AbsRotateCoordinateSystem(protate)

' Set chart depth
chartVu.SetFractionalZViewportDepth(0.1)

.
.
.

' Define a simple line marker plot
Dim attrib1 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid)
Dim attrib2 As New ChartAttribute(Color.Red, 1, DashStyle.Solid)
attrib2.SetFillColor(Color.Red)
attrib2.SetFillFlag(True)
attrib2.SetSymbolSize(15)
Dim thePlot1 As New SimpleLineMarkerPlot(pTransform1, Dataset1, _
    ChartObj.CUBE3D, attrib1, attrib2, 0, 1)
chartVu.AddChartObject(thePlot1)

```

Add the following lines to the program segment above to add data point labeling to the line marker plot.

[C#]

```

// Customize display of plot data point values
thePlot1.SetShowDatapointValue(true);
NumericLabel modellabel = new NumericLabel();
modellabel.SetXJust(ChartObj.JUSTIFY_CENTER);
modellabel.SetYJust(ChartObj.JUSTIFY_MIN);
Font modellabelfont = new Font("Microsoft Sans Serif", 10, FontStyle.Regular);
modellabel.SetTextFont(modellabelfont);
modellabel.SetTextNudge(0, -5);
thePlot1.SetPlotLabelTemplate(modellabel);

```

[Visual Basic]

```

' Customize display of plot data point values
thePlot1.SetShowDatapointValue(True)
Dim modellabel As New NumericLabel()

```



```
modellabel.SetXJust(ChartObj.JUSTIFY_CENTER)
modellabel.SetYJust(ChartObj.JUSTIFY_MIN)
Dim modellabelfont As New Font("Microsoft Sans Serif", 10, FontStyle.Regular)
modellabel.SetTextFont(modellabelfont)
modellabel.SetTextNudge(0, -5)
thePlot1.SetPlotLabelTemplate(modellabel)
```


11. Group Plot Objects

GroupPlot

- ArrowPlot
- BubblePlot
- CandlestickPlot
- CellPlot
- ErrorBarPlot
- FloatingBarPlot
- FloatingStackedBar
- GroupBarPlotChartPlot
- HistogramPlot
- LineGapPlot
- MultiLinePlot
- OHLCPlot
- OHLCBarPlot
- StackedBarPlot
- StackedLinePlot

The **GroupPlot** class is an abstract class representing plot types that use data organized as arrays of x- and y-values, where there is one or more y-value for each x-value. Group plot types include: multi-line plots, stacked line plots, stacked bar plots, group bar plots, error bar plots, floating bar plots, open-high-low-close plots, candlestick plots, arrow plots, histogram plots, cell plots and bubble plots.

The number of x-values in a group plot is referred to as the number of columns, or as **numberDatapoints** and the number of y-values *for each x-value* is referred to as the number of rows, or **numberGroups**. Think of spreadsheet that looks like:

x-values	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
y-values group #0	y[0,0]	y[0,1]	y[0,2]	y[0,3]	y[0,4]	y[0,5]
y-values group #1	y[1,0]	y[1,1]	y[1,2]	y[1,3]	y[1,4]	y[1,5]
y-values group #2	y[2,0]	y[2,1]	y[2,2]	y[2,3]	y[2,4]	y[2,5]

number of x-values = **numberDatapoints** = numberColumns = 6

number of y-values for each x-value = **numberGroups** = numberRows = 3

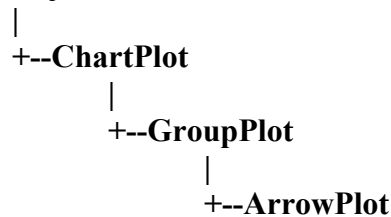
This would be the ROW_MAJOR format if the data were stored in a CSV file.

Example program segments presented in this documentation are not complete programs and contain uninitialized and/or undefined objects and variables. Do not attempt to copy them into your own program. Refer to the referenced example program that the code is extracted from.

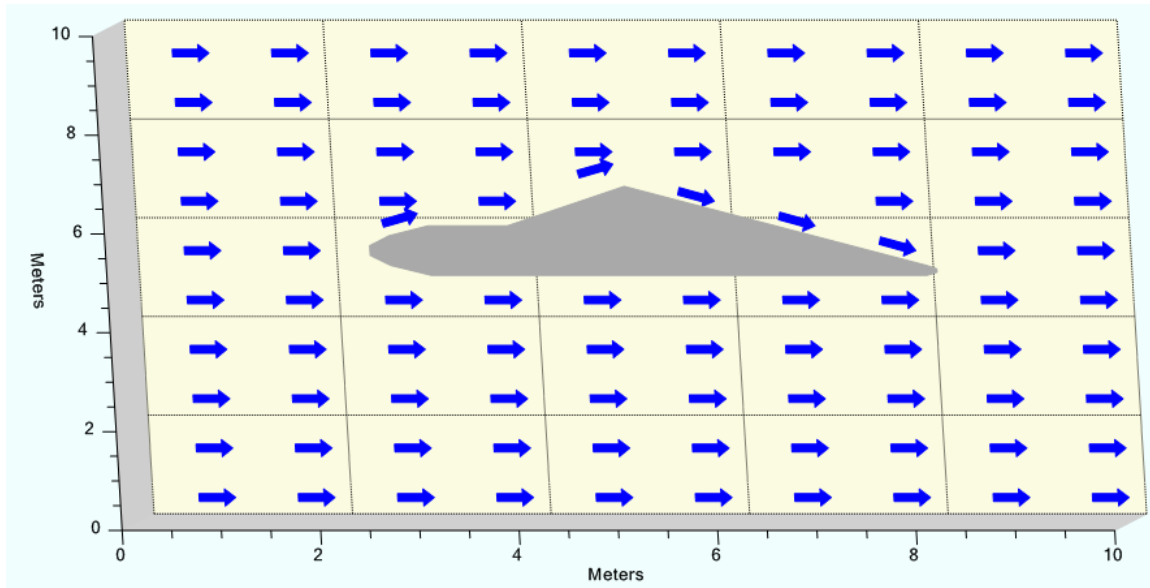
Arrow Plots

Class ArrowPlot

GraphObj



The **ArrowPlot** class is a subclass of the **GroupPlot** class. It displays a collection of arrows as defined by the data in a group dataset. The position, size, and rotation of each arrow in the collection is independently controlled. The number of groups of the group dataset must be three. When used with a 2½ D dataset, the position of the each arrow with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property is not used. When an arrow plot is attached to a 3D dataset, each arrow is not restricted to a z-axis plane and can traverse 3D space unrestricted.



Arrow plot (ScatterPlots.ArrowChart)

ArrowPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal basearrow As Arrow, _
    ByVal attrib As ChartAttribute _
)

[C#]
public ArrowPlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    Arrow basearrow,
    ChartAttribute attrib
);
```

transform

The coordinate system for the new **ArrowPlot** object.

dataset

The group dataset sets the position, size and rotation of individual arrows. The number of groups must be three. Organize the data in the dataset in the following manner:

X x-position of the arrow point.

Y[0] y-position of the arrow point.

Y[1] Size of the arrow. A size of 0.05 creates an arrow with a length equal to 0.05 in NORM_PLOT_POS coordinates.

Y[2] The rotation of the arrow, using the point of the arrow as the rotation origin, in degrees.

Z z-position of the arrow point.

basearrow An instance of an **Arrow** object used to draw the arrows in this **ArrowPlot** object.

attrib Sets the color, line and fill characteristics for the arrows in this **ArrowPlot** object.

An individual arrow in an arrow plot object can have unique attributes. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods in the manner described for **SimplePlot** objects.

Arrow plot example (extracted from the example program ScatterPlots.ArrowChart)

[C#]

```
// Create a group dataset
GroupDataset Dataset1 = new GroupDataset("First",x1,y1);
Dataset1.ImplicitZValue = 0.5;
// only auto-scale using the first group of the group data
Dataset1.SetAutoScaleNumberGroups(1);
// Create a coordinate system that matches the data
CartesianCoordinates pTransform1 =
new CartesianCoordinates( ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
// Force x- and y-scales to start at 0
pTransform1.SetScaleX(0,10);
pTransform1.SetScaleY(0,10);
// Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.125, .15, .90, 0.8) ;

// Set rotational parameters
Point3D protate = new Point3D(10, 20, 0);
pTransform1.AbsRotateCoordinateSystem( protate);
// Set chart depth
chartVu.SetFractionalZViewportDepth(0.1);
.
.
```

```

.
// Define the arrow plot
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid);
attrib1.SetFillColor (Color.Blue);
attrib1.SetFillFlag (true);
Arrow basearrow = new Arrow();
ArrowPlot thePlot1 = new ArrowPlot(pTransform1, Dataset1, basearrow, attrib1);
chartVu.AddChartObject(thePlot1);

```

[Visual Basic]

```

Dim Dataset1 As New GroupDataset("First", x1, y1)
Dataset1.ImplicitZValue = 0.5
' only auto-scale using the first group of the group data
Dataset1.SetAutoScaleNumberGroups(1)
' Create a coordinate sysetm that matches the data
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE,
ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
' Force x- and y-scales to start at 0
pTransform1.SetScaleX(0, 10)
pTransform1.SetScaleY(0, 10)
' Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.125, 0.15, 0.9, 0.8)

' Set rotational parameters
Dim protate As New Point3D(10, 20, 0)
pTransform1.AbsRotateCoordinateSystem(protate)
' Set chart depth
chartVu.SetFractionalZViewportDepth(0.1)
.
.
.
' Define the arrow plot
Dim attrib1 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid)
attrib1.SetFillColor(Color.Blue)
attrib1.SetFillFlag(True)
Dim basearrow As New Arrow()
Dim thePlot1 As New ArrowPlot(pTransform1, Dataset1, basearrow, attrib1)
chartVu.AddChartObject(thePlot1)

```

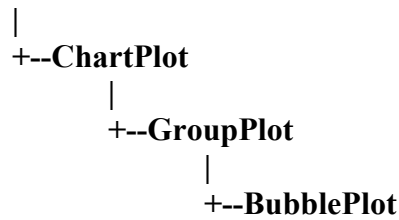
*Note the use of the **GroupDataset** method **SetAutoScaleNumberGroups**. This forces the auto-scale routine to look at only the first group of y-values, since those are the only

y-values that specify the absolute position of the arrows. The other groups of y-values specify size and rotation information and should not be considered in the auto-scale calculation.

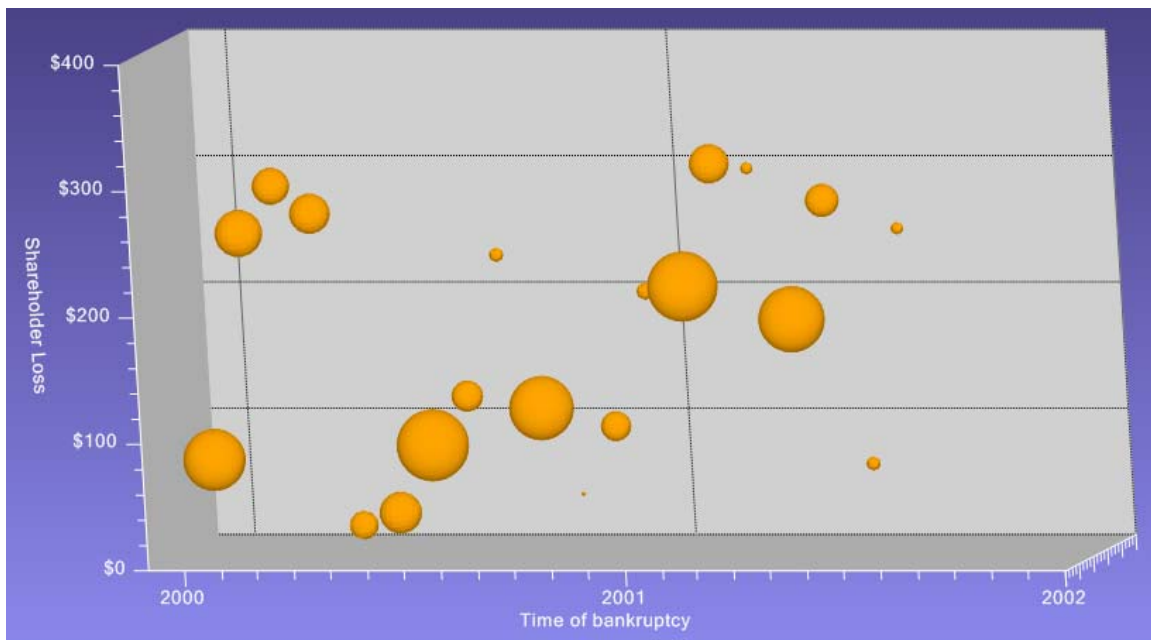
Bubble Plots

Class BubblePlot

GraphObj



The **BubblePlot** class is a subclass of the **GroupPlot** class. It displays bubble plots. A group dataset specifies the position and size of each bubble in a bubble plot. The number of groups must be two. When used with a 2½ D dataset, the position of the each bubble with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property is not used. When a bubble plot is attached to a 3D dataset, each bubble is not restricted to a z-axis plane and can traverse 3D space unrestricted. A bubble can take one of two forms: the first is a disk shaped object; the second is a sphere shaped object.



Bubble plot: sphere mode. (ScatterPlots.BubbleChart)

BubblePlot constructor

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal bubblesizetype As Integer, _
    ByVal attrib As ChartAttribute _
)

[C#]
public BubblePlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    int bubblesizetype,
    ChartAttribute attrib
);

```

<i>transform</i>	The coordinate system for the new bubble plot object.
<i>dataset</i>	A group dataset specifying the location and size of the bubbles in the bubble plot. The number of groups must be two. The dataset values for X and Y[0] set the position of the center of each bubble and the values for Y[1] set the size of each bubble, either the area (SIZE_BUBBLE_AREA) or the radius(SIZE_BUBBLE_RADIUS).
<i>bubblesizetype</i>	Sets whether the circle representing each bubble plot has a radius, or an area, proportional to the Y[1] data values in the group dataset. Set using one of the bubble plot type constants: SIZE_BUBBLE_RADIUS or SIZE_BUBBLE_AREA.
<i>attrib</i>	Specifies the attributes (line color and fill color) of the bubble plot circles.

A bubble is by default rendered as a sphere. You can explicitly set the rendering mode using the `BubblePlotMode` property, setting it to either `SPHERE3D` or `CYLINDER3D`.

An individual bubble in a bubble plot object can have unique attributes. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods in the manner described for **SimplePlot** objects.

**Bubble plot example (extracted from the example program
ScatterPlots.BubbleChart)**

```

[C#]

```

254 Group Plot Objects

```
// Create dataset
TimeGroupDataset Dataset1 = new TimeGroupDataset("First",x1,y1);
// By adding the y1[0, index] value (position) to the y1[1, index] (radius) value,
we
// should get an auto-axis scale that takes into account the bubble radius
Dataset1.SetStackMode(CharObj.AUTOAXES_STACKED);
Dataset1.ImplicitZValue = 0.5;

// Define coordinate sysetm for data
TimeCoordinates pTransform1 = new TimeCoordinates( CharObj.TIME_SCALE,
CharObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, CharObj.AUTOAXES_FAR, CharObj.AUTOAXES_FAR);
// Move starting date back one month to allow for radius of bubble
.
.
.
// Define a bubble plot
ChartAttribute attrib1 = new ChartAttribute (Color.Orange, 0,DashStyle.Solid,
Color.Orange);
BubblePlot thePlot1 = new BubblePlot(pTransform1, Dataset1,
CharObj.SIZE_BUBBLE_RADIUS, attrib1);
thePlot1.ImplicitDepthValue = 0.05;
thePlot1.BubblePlotMode = CharObj.SPHERE3D;
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
' Create dataset
Dim Dataset1 As New TimeGroupDataset("First", x1, y1)
' By adding the y1[0, index] value (position) to the y1[1, index] (radius) value,
we
' should get an auto-axis scale that takes into account the bubble radius
Dataset1.SetStackMode(CharObj.AUTOAXES_STACKED)
Dataset1.ImplicitZValue = 0.5

' Define coordinate sysetm for data
Dim pTransform1 As New TimeCoordinates(CharObj.TIME_SCALE, CharObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, CharObj.AUTOAXES_FAR, CharObj.AUTOAXES_FAR)
.
.
.
' Define a bubble plot
Dim attrib1 As New ChartAttribute(Color.Orange, 0, DashStyle.Solid, Color.Orange)
Dim thePlot1 As New BubblePlot(pTransform1, Dataset1, _
```

```

    ChartObj.SIZE_BUBBLE_RADIUS, attrib1)
thePlot1.ImplicitDepthValue = 0.05
thePlot1.BubblePlotMode = ChartObj.SPHERE3D
chartVu.AddChartObject(thePlot1)

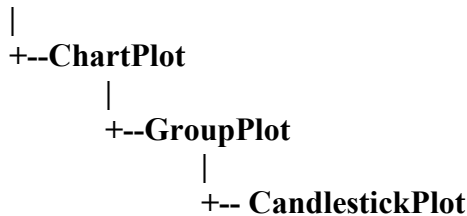
```

*Note the use of the **GroupDataset** method **SetStackMode**. This forces the auto-scale routine to look at the sum of y-values across groups, as is needed to auto-scale stacked plots. It is useful for bubble plots of type **SIZE_BUBBLE_RADIUS** because the `y[0]` value represents the y-position of the bubble, and the `y[1]` value the radius in physical coordinates. Adding the two for each bubble gives the maximum y-value for the scale needed to display the bubble. If **SIZE_BUBBLE_AREA** is used you may want to restrict the auto-scale routines to the just look at the bubble position using **SetAutoScaleNumberGroups(1)**, as seen in the **ArrowPlot** example. You could then add in some fudge factor to make sure that the scale shows the entire bubble. The example under **CellPlot** demonstrates this.

Candlestick Plots

Class CandlestickPlot

GraphObj



The **CandlestickPlot** class is a subclass of the **GroupPlot** class. It extends the **GroupPlot** class and displays stock market data in an open-high-low-close format common in financial technical analysis. Every item of the plot is a group of two horizontal lines representing High and Low values which are connected with a vertical line and a box representing the Open and Close values. If the Open value is greater than the Close value for a particular candlestick, the box is filled, otherwise it is unfilled. The number of groups must be four. The data in the dataset is organized in the following manner: The `Y[0]` values of the group dataset represent the values for Open, the `Y[1]` values for High, the `Y[2]` values for Low, and the `Y[3]` values for Close. When used with a 2½ D dataset, the position of the each candlestick symbol with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property specifies the 3D depth of the candlestick box. When a candlestick plot is attached to a 3D dataset, each candlestick symbol is not restricted to a z-axis plane and can traverse 3D space unrestricted.



Candlestick plot (FinancialExamples.CandlestickChart)

CandlestickPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal rwidth As Double, _
    ByVal defaultattrib As ChartAttribute, _
    ByVal fillattrib As ChartAttribute _
)

[C#]
public CandlestickPlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    double rwidth,
    ChartAttribute defaultattrib,
    ChartAttribute fillattrib
);
```

transform

The coordinate system for the new **CandlestickPlot** object.

dataset

The **CandlestickPlot** plot represents the group open-high-low-close values in this group dataset. The number of groups must be four. Organize the data in the following manner: The x-values of the group dataset set the x-positions of the candlestick objects. The Y[0] values of the group dataset represent the values for Open, the Y[1] values for High, the Y[2] values for Low, and the Y[3] values for Close.

<i>rwidth</i>	The width of the candlestick box in physical coordinates.
<i>defaultattrib</i>	Specifies the default attributes (line color and fill color) of the candlestick lines and box.
<i>fillattrib</i>	Specifies the attributes (line color and fill color) of the candlestick lines when the close value is greater than the open value.

An individual candlestick in a candlestick plot object can have unique attributes. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods in the manner described for **SimplePlot** objects.

Candlestick plot example (extracted from the example program **FinancialExamples.CandlestickChart**)

[C#]

```
// Create a group dataset to hold stock data
double zvalue = 0.0;
double depthvalue = 0.1;
TimeGroupDataset Dataset1 = new TimeGroupDataset("Stock
Data",xValues,stockPriceData, zvalue, depthvalue);

// Define a coordinate system to hold data
pTransform1 = new TimeCoordinates();
// Set coordinate system for 5 day week
pTransform1.SetWeekType(ChartObj.WEEK_5D);
pTransform1.AutoScale(Dataset1,ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR);
// Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.13, .15, .90, 0.8) ;

// Set rotational parameters
Point3D protate = new Point3D(10, 10, 0);
pTransform1.AbsRotateCoordinateSystem( protate);
// Set chart depth
chartVu.SetFractionalZViewportDepth(0.1);
.
.
.
// Define a candlestick plot
ChartAttribute defaultattrib = new ChartAttribute(Color.Black, 1,DashStyle.Solid,
Color.White);
defaultattrib.SetFillFlag(true);
```

258 *Group Plot Objects*

```
ChartAttribute fillattrib = new ChartAttribute(Color.Black, 1, DashStyle.Solid,
Color.Red);
fillattrib.SetFillFlag(true);
CandlestickPlot thePlot1 =
    new CandlestickPlot(pTransform1, Dataset1,
        ChartCalendar.GetCalendarWidthValue(ChartObj.DAY_OF_YEAR, 0.8),
        defaultattrib, fillattrib);
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
' Create a group dataset to hold stock data
Dim zvalue As Double = 0.0
Dim depthvalue As Double = 0.1
Dim Dataset1 As New TimeGroupDataset("Stock Data", xValues, _
    stockPriceData, zvalue, depthvalue)

' Define a coordinate system to hold data
Dim pTransform1 As New TimeCoordinates()
' Set coordinate system for 5 day week
pTransform1.SetWeekType(ChartObj.WEEK_5D)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR)
' Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.13, 0.15, 0.9, 0.8)

' Set rotational parameters
Dim protate As New Point3D(10, 10, 0)
pTransform1.AbsRotateCoordinateSystem(protate)
' Set chart depth
chartVu.SetFractionalZViewportDepth(0.1)
.
.
.
' Define a candlestick plot
Dim defaultattrib As New ChartAttribute(Color.Black, 1, _
    DashStyle.Solid, Color.White)
defaultattrib.SetFillFlag(True)
Dim fillattrib As New ChartAttribute(Color.Black, 1, DashStyle.Solid, _
    Color.Red)
fillattrib.SetFillFlag(True)
Dim thePlot1 As New CandlestickPlot(pTransform1, Dataset1, _
    ChartCalendar.GetCalendarWidthValue(ChartObj.DAY_OF_YEAR, 0.8), _
    defaultattrib, fillattrib)
```

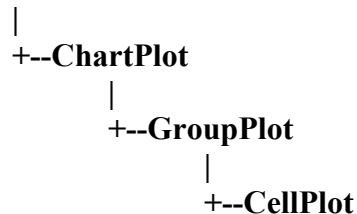
```
chartVu.AddChartObject(thePlot1)
```

* Note how the **ChartCalendar.GetCalendarWidthValue** method calculates the width of the bars as a function of time, in this case a width of 0.8 months.

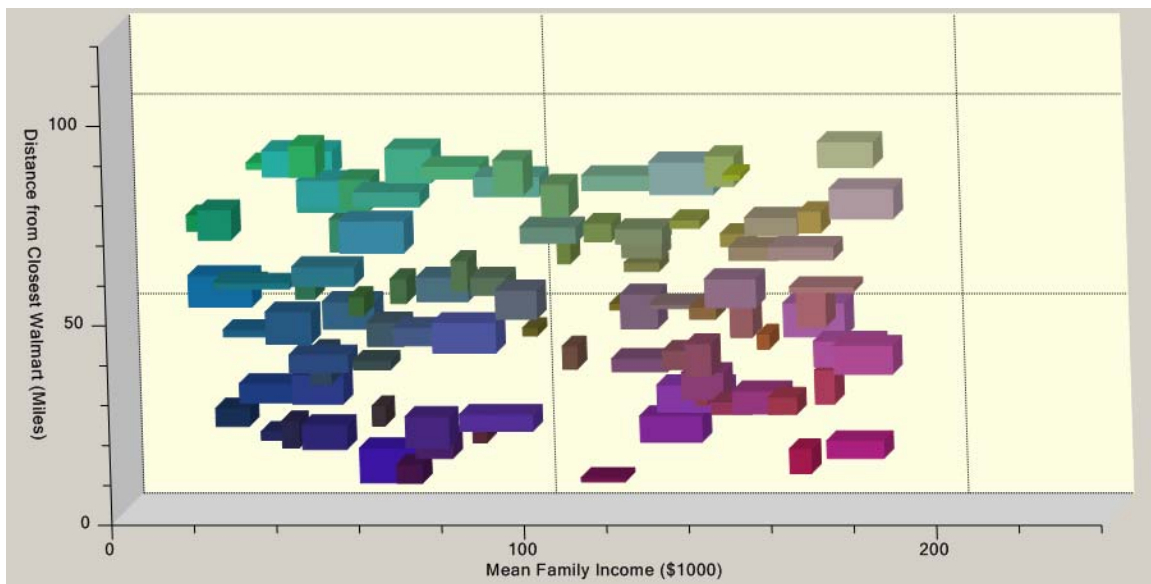
Cell Plots

Class CellPlot

GraphObj



The **CellPlot** class extends the **GroupPlot** class and displays cell plots. A cell plot is a collection of rectangular objects with independent positions, widths and heights, specified using the values of the associated group dataset. The number of groups must be three. The (X, Y[0]) values of the group dataset represent the xy position of the lower left corner of each cell, the Y[1] values set the width of the cell, and the Y[2] values set the height of the cell. Each cell can be filled using a color, or an image. When used with a 2½ D dataset, the position of the each cell with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property specifies the 3D depth of the cell box. When a cell plot is attached to a 3D dataset, each cell box is not restricted to a z-axis plane and can traverse 3D space unrestricted.



*Cell plot (ScatterPlots.CellPlotChart)***CellPlot constructor**

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal attrib As ChartAttribute _
)

[C#]
public CellPlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    ChartAttribute attrib
);

```

<i>transform</i>	The coordinate system for the new CellPlot object.
<i>dataset</i>	The cell plot represents the values in this group dataset. The number of groups must be three. The (X, Y[0]) values of the group dataset represent the xy position of the lower left corner of each cell, the Y[1] values set the width of the cell, and the Y[2] values set the height of the cell.
<i>attrib</i>	Specifies the attributes (line color and line style) for the cell plot.

Cells can be filled with an image instead of a solid color. Use the **CellPlot.SetPlotImage** method to place a System.Drawing.Image object in the cells of a cell plot. One image applies to all of the cells in the cell plot.

An individual cell in the cell plot object can have unique attributes. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods in the manner described for **SimplePlot** objects.

Cell plot example (extracted from the example program ScatterPlots.CellPlotChart)

```

[C#]

// Create dataset
GroupDataset Dataset1 = new GroupDataset("First",x1,y1);
Dataset1.ImplicitDepthValue = 0.3;

```



```

Dataset1.SetAutoScaleNumberGroups(1); // picks up on width, but because data is
random, should still work

CartesianCoordinates pTransform1 = new CartesianCoordinates(
ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);

pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
// Add-in max width of cells
double maxx = pTransform1.GetScaleStopX() + 20;
// Add-in max height of cells
double maxy = pTransform1.GetScaleStopY() + 10;
pTransform1.SetScaleStopX(maxx);
pTransform1.SetScaleStopY(maxy);
// Re-auto-scale to produce rounded axis values.
pTransform1.AutoScale(ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR);

// position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, .15, .90, 0.8) ;

// Set rotational parameters
Point3D protate = new Point3D(10, 10, 0);
pTransform1.AbsRotateCoordinateSystem( protate);
// Set chart depth
chartVu.SetFractionalZViewportDepth(0.2);
.
.
.
// Define a cell plot
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid);
attrib1.SetFillColor (Color.Blue);
attrib1.SetFillFlag (true);
CellPlot thePlot1 = new CellPlot(pTransform1, Dataset1, attrib1);
thePlot1.ImplicitDepthValue = 0.3;
for (i=0; i < numPoints; i++)
    thePlot1.SetSegmentColor(i, Color.FromArgb((int) (x1[i]),
        (int) (y1[0,i]*2.0),
        (int) ((y1[1,i] + y1[2,i])* 7)));
chartVu.AddChartObject(thePlot1);

```

[Visual Basic]

```

' Create dataset
Dim Dataset1 As New GroupDataset("First", x1, y1)
Dataset1.ImplicitDepthValue = 0.3
Dataset1.SetAutoScaleNumberGroups(1)
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _

```

262 Group Plot Objects

```
ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
' Add-in max width of cells
Dim maxx As Double = pTransform1.GetScaleStopX() + 20
' Add-in max height of cells
Dim maxy As Double = pTransform1.GetScaleStopY() + 10
pTransform1.SetScaleStopX(maxxx)
pTransform1.SetScaleStopY(maxy)
' Re-auto-scale to produce rounded axis values.
pTransform1.AutoScale(ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR)

' position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 0.8)

' Set rotational parameters
Dim protate As New Point3D(10, 10, 0)
pTransform1.AbsRotateCoordinateSystem(protate)
' Set chart depth
chartVu.SetFractionalZViewportDepth(0.2)

.
.
.

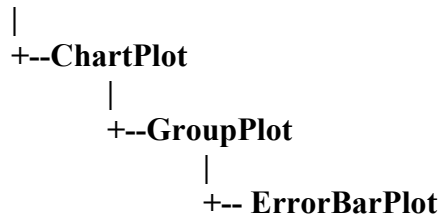
' Define a cell plot
Dim attrib1 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid)
attrib1.SetFillColor(Color.Blue)
attrib1.SetFillFlag(True)
Dim thePlot1 As New CellPlot(pTransform1, Dataset1, attrib1)
thePlot1.ImplicitDepthValue = 0.3
For i = 0 To numPoints - 1
    thePlot1.SetSegmentColor(i, Color.FromArgb(CInt(x1(i)), _
        CInt(y1(0, i) * 2.0), CInt((y1(1, i) + y1(2, i)) * 7)))
Next i
chartVu.AddChartObject(thePlot1)
```

*Note the use of the **GroupDataset** method **SetAutoScaleNumberGroups**. This forces the auto-scale routine to look at just the first group of values, Y[0], because those are the only absolute position values. The maximum cell width and height are calculated and added to the initial scale. The auto-scale function is then rerun, producing a coordinate system that takes into account the widths and heights of the cells.

Error Bar Plots

Class ErrorBarPlot

GraphObj



The **ErrorBarPlot** class extends the **GroupPlot** class and displays error bars. Error bars are two lines positioned around a data point to signify the statistical error associated with the data point. The number of groups must be two. The (X, Y[0]) values of the group dataset represent the xy position of the first error bar lines, the (X, Y[1]) values of the group dataset represent the xy position of the second error bar lines. The error bar lines center on the X. Connecting the error bar lines with a perpendicular line is an option. . When used with a 2½ D dataset, the position of the each error bar with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property specifies the 3D depth of the error bar planes. When a error bar plot is attached to a 3D dataset, each cell plane is not restricted to a z-axis plane and can traverse 3D space unrestricted.

ErrorBarPlot constructor

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal rbarwidth As Double, _
    ByVal attrib As ChartAttribute _
)

[C#]
public ErrorBarPlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    double rbarwidth,
    ChartAttribute attrib
);
  
```

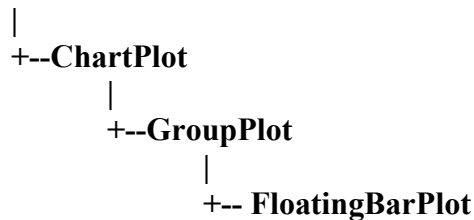
<i>transform</i>	The coordinate system for the new ErrorBarPlot object. The number of groups must be two.
<i>dataset</i>	The error bar plot represents the values in this group dataset. The number of groups must be two. The (X, Y[0]) values of the group dataset represent the xy position of the first error bar lines, the (X, Y[1]) values of the group dataset represent the xy position of the second error bar lines.
<i>rbarwidth</i>	The width of the error bars.
<i>attrib</i>	Specifies the attributes (line color and line style) for the error bars.

An individual set of error bars in an error bar plot object can have unique attributes. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods in the manner described for **SimplePlot** objects.

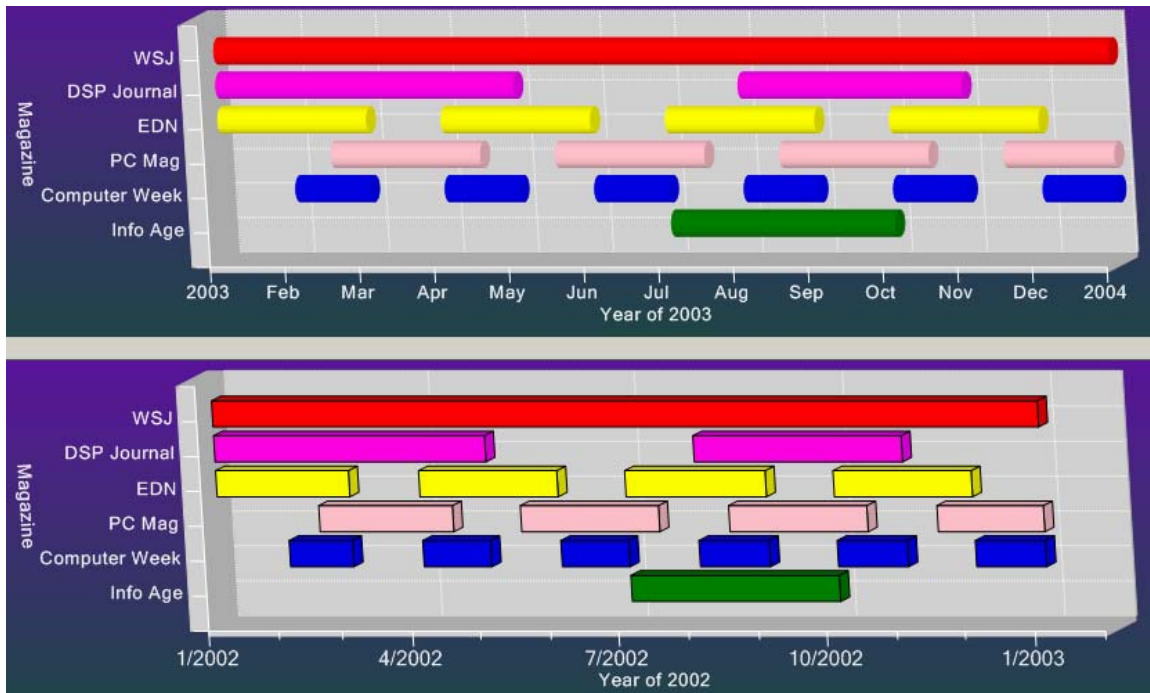
Floating Bar Plots

Class FloatingBarPlot

GraphObj



The **FloatingBarPlot** class extends the **GroupPlot** class and displays floating bar plots. The bars are free floating because each bar does not reference a fixed base value, as do the simple bar plots, stacked bar plots and group bar plots. The number of groups must be two. The (X, Y[0]) values of the group dataset represent the starting points of each bar, the (X, Y[1]) values of the group dataset represent the ending points of each bar. All bars in a given **FloatingBarPlot** object have the same width. When used with a 2½ D dataset, the position of the each bar with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property specifies the 3D depth of the bar. When a floating bar plot is attached to a 3D dataset, each bar is not restricted to a z-axis plane and can traverse 3D space unrestricted. The individual elements of the bar plot are normally 3D bars, but you can also set the bar plots **BarShapeOption** property to a cylinder (CYLINDER3D).



Floating bar plots (Bargraphs.FloatingBars)

FloatingBarPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal rbarwidth As Double, _
    ByVal attrib As ChartAttribute, _
    ByVal nbarjust As Integer _
)

[C#]
public FloatingBarPlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    double rbarwidth,
    ChartAttribute attrib,
    int nbarjust
);
```

transform

The coordinate system for the new **FloatingBarPlot** object.

dataset

The floating bar plot represents the values in this group dataset. The number of groups must be two. The (X, Y[0]) values of the group dataset represent the starting points of each bar, the (X, Y[1]) values of the group dataset represent the ending points of each bar.

rbarwidth

The width of the floating bars in units of the independent axis.

<i>attrib</i>	Specifies the attributes (line and fill color) for the floating bars.
<i>nbarjust</i>	Specifies the justification with respect to the independent data value. Use one of the justification constants: JUSTIFY_MIN, JUSTIFY_CENTER, JUSTIFY_MAX.

An individual bar in a floating bar plot object can have unique attributes. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods in the manner described for **SimplePlot** objects.

Scheduling charts often use floating bars, where the starting and ending values of the bar represent the duration of some aspect of a project. The default use of the floating bar class assumes that the ends of the bars are floating point values, not date/time values. Yet the scheduling chart often uses date/time values to specify the bar ends. Since only the x-axis works with time values, the floating bars of a scheduling chart need to be used in the horizontal orientation mode, set using the **FloatingBar.SetBarOrient** method. Used in this mode, the x-values of the dataset position the bars with respect to the y-axis, and the y-values of the dataset position the ends of the bars with respect to the x-axis. In order to have a group dataset object that stores x-values as doubles and the y-group values as **ChartCalendar** dates you must use a special **TimeGroupDataset** constructor:

```
[Visual Basic]
[Visual Basic]
Overloads Public Sub New( _
    ByVal sname As String, _
    ByVal x As ChartCalendar(), _
    ByVal y As ChartCalendar(), _
)

[C#]
public TimeGroupDataset(
    string sname,
    ChartCalendar[] x,
    ChartCalendar[,] y
);
```

If you manually scale the **TimeCoordinates** object, you can proceed as in all the other examples that use a date/time x-axis. If you need to use the auto-scaling capability, you will need to add a few additional steps.

Place the data in a **TimeGroupDataset** dataset and use it to auto-scale a **TimeCoordinates** scaling object. The only problem here is that since the y-values are the time values, the chart y-axis will end up as the time axis and the x-axis will end up the numeric axis. Call the **TimeCoordinates.SwapScaleOrientation** in order to get it back to the orientation we want. This swaps the scale orientation so the x-axis is the time axis and the y-axis is the numeric axis. See the second of the two examples below for more details about how to use date/time values to specify the bar ends of a floating bar plot.

**Floating bar plot example that uses numeric values as the floating bar endpoints
(extracted from the example program Bargraphs.FloatingBars)**

[C#]

```
int numpnts = 18;
int numgroups = 2;
double []x1= new double[numpnts];
double [,]y1 = new double[numgroups,numpnts];

x1[0] = 6;
y1[0,0] = 0;
y1[1,0] = 12;

x1[1] = 5;
y1[0,1] = 0;
y1[1,1] = 4;

x1[2] = 5;
y1[0,2] = 7;
y1[1,2] = 10;

x1[3] = 4;
y1[0,3] = 0;
y1[1,3] = 2;
.
.
.
GroupDataset Dataset1 = new GroupDataset("Actual Sales",x1,y1);
Dataset1.ImplicitDepthValue = 0.1;
Dataset1.ImplicitZValue = 0.5;

pTransform1 = new CartesianCoordinates();

pTransform1.SetScaleStartX(0);
pTransform1.SetScaleStartY(0);

pTransform1.SetScaleStopX(12);
pTransform1.SetScaleStopY(7);

pTransform1.SetGraphBorderDiagonal(0.2, .15, .95, 0.8) ;
Background graphbackground = new Background( pTransform1,
ChartObj.GRAPH_BACKGROUND,
```

268 *Group Plot Objects*

```
Color.FromArgb(30,70,70), Color.FromArgb(90,20,155), ChartObj.Y_AXIS);
chartVu.AddChartObject(graphbackground);

// Set chart rotation
Point3D protate = new Point3D(10, 20, 0);
pTransform1.AbsRotateCoordinateSystem(protate);
chartVu.SetFractionalZViewportDepth(0.1);

.
.
.

ChartAttribute attrib1 = new ChartAttribute (Color.Black, 1,DashStyle.Solid,
Color.Green);
attrib1.SetFillFlag(true);
attrib1.LineFlag = true;

FloatingBarPlot thePlot1 = new FloatingBarPlot(pTransform1, Dataset1, 0.75,
attrib1, ChartObj.JUSTIFY_CENTER);
thePlot1.SetBarOrient(ChartObj.HORIZ_DIR);
thePlot1.BarShapeOption = ChartObj.CYLINDER3D;
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
Dim numpnts As Integer = 18
Dim numgroups As Integer = 2
Dim x1(numpnts - 1) As Double
Dim y1(numgroups - 1, numpnts - 1) As Double

x1(0) = 6
y1(0, 0) = 0
y1(1, 0) = 12

x1(1) = 5
y1(0, 1) = 0
y1(1, 1) = 4

x1(2) = 5
y1(0, 2) = 7
y1(1, 2) = 10

x1(3) = 4
y1(0, 3) = 0
y1(1, 3) = 2

.
```



```

.
.

Dim Dataset1 As New GroupDataset("Actual Sales", x1, y1)
Dataset1.ImplicitDepthValue = 0.1
Dataset1.ImplicitZValue = 0.5

pTransform1 = New CartesianCoordinates()

pTransform1.SetScaleStartX(0)
pTransform1.SetScaleStartY(0)

pTransform1.SetScaleStopX(12)
pTransform1.SetScaleStopY(7)

pTransform1.SetGraphBorderDiagonal(0.2, 0.15, 0.95, 0.8)
Dim graphbackground As New Background(pTransform1, ChartObj.GRAPH_BACKGROUND,
Color.FromArgb(30, 70, 70), Color.FromArgb(90, 20, 155), ChartObj.Y_AXIS)
chartVu.AddChartObject(graphbackground)

' Set chart rotation
Dim protate As New Point3D(10, 20, 0)
pTransform1.AbsRotateCoordinateSystem(protate)
chartVu.SetFractionalZViewportDepth(0.1)

.
.
.

Dim attrib1 As New ChartAttribute(Color.Black, 1, DashStyle.Solid, Color.Green)
attrib1.SetFillFlag(True)
attrib1.LineFlag = True

Dim thePlot1 As New FloatingBarPlot(pTransform1, Dataset1, 0.75, attrib1,
ChartObj.JUSTIFY_CENTER)
thePlot1.SetBarOrient(ChartObj.HORIZ_DIR)
thePlot1.BarShapeOption = ChartObj.CYLINDER3D
chartVu.AddChartObject(thePlot1)

```

**Floating bar plot example that uses date/time values as the floating bar endpoints
(extracted from the example program Bargraphs.FloatingBars2)**

[C#]

```
int nnumpts = 18;
```

270 *Group Plot Objects*

```
int numgroups = 2;
double []x1= new double[numpnts];
ChartCalendar [,]y1 = new ChartCalendar[numgroups,numpnts];

x1[0] = 6;
y1[0,0] = new ChartCalendar(2002, ChartObj.JANUARY,1);
y1[1,0] = new ChartCalendar(2003, ChartObj.JANUARY,1);

x1[1] = 5;
y1[0,1] = new ChartCalendar(2002, ChartObj.JANUARY,1);
y1[1,1] = new ChartCalendar(2002, ChartObj.MAY,1);
.
.
.

TimeGroupDataset Dataset1 = new TimeGroupDataset("Actual Sales",x1,y1, 0.1, 0.8);
Dataset1.ImplicitDepthValue = 0.3;
Dataset1.ImplicitZValue = 0.5;

TimeCoordinates pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(Dataset1,ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_NEAR);

pTransform1.SwapScaleOrientation();
pTransform1.SetScaleStartY(0);
pTransform1.SetGraphBorderDiagonal(0.2, .05, .95, 0.75) ;

Background graphbackground = new Background( pTransform1,
ChartObj.GRAPH_BACKGROUND,Color.FromArgb(30,70,70), Color.FromArgb(90,20,155),
ChartObj.Y_AXIS);
chartVu.AddChartObject(graphbackground);

// Set chart rotation
Point3D protate = new Point3D(10, 20, 0);
pTransform1.AbsRotateCoordinateSystem(protate);
chartVu.SetFractionalZViewportDepth(0.1);

.
.
.

FloatingBarPlot thePlot1 = new FloatingBarPlot(pTransform1);
ChartAttribute attrib1 = new ChartAttribute (Color.Black, 1,DashStyle.Solid,
Color.Green);
attrib1.SetFillFlag(true);
thePlot1.InitFloatingBarPlot(Dataset1, 0.75, attrib1, ChartObj.JUSTIFY_CENTER);
```

```
thePlot1.SetBarOrient(ChartObj.HORIZ_DIR);
```

```
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
Dim numpnts As Integer = 18
Dim numgroups As Integer = 2
Dim x1(numpnts - 1) As Double
Dim y1(numgroups - 1, numpnts - 1) As ChartCalendar

x1(0) = 6
y1(0, 0) = New ChartCalendar(2002, ChartObj.JANUARY, 1)
y1(1, 0) = New ChartCalendar(2003, ChartObj.JANUARY, 1)

x1(1) = 5
y1(0, 1) = New ChartCalendar(2002, ChartObj.JANUARY, 1)
y1(1, 1) = New ChartCalendar(2002, ChartObj.MAY, 1)

x1(2) = 5
y1(0, 2) = New ChartCalendar(2002, ChartObj.AUGUST, 1)
y1(1, 2) = New ChartCalendar(2002, ChartObj.NOVEMBER, 1)

x1(3) = 4
y1(0, 3) = New ChartCalendar(2002, ChartObj.JANUARY, 1)
y1(1, 3) = New ChartCalendar(2002, ChartObj.MARCH, 1)

.
.
.

Dim Dataset1 As New TimeGroupDataset("Actual Sales", x1, y1, 0.1, 0.8)
Dataset1.ImplicitDepthValue = 0.3
Dataset1.ImplicitZValue = 0.5

Dim pTransform1 As New TimeCoordinates()
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_NEAR)

pTransform1.SwapScaleOrientation()
pTransform1.SetScaleStartY(0)
pTransform1.SetGraphBorderDiagonal(0.2, 0.05, 0.95, 0.75)

Dim graphbackground As New Background(pTransform1, ChartObj.GRAPH_BACKGROUND,
Color.FromArgb(30, 70, 70), Color.FromArgb(90, 20, 155), ChartObj.Y_AXIS)
chartVu.AddChartObject(graphbackground)
```

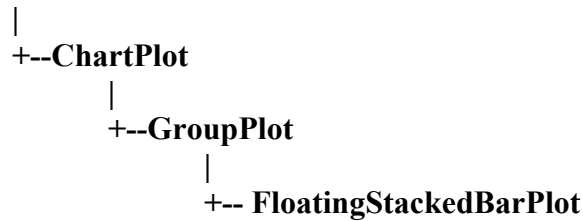
```

' Set chart rotation
Dim protate As New Point3D(10, 20, 0)
pTransform1.AbsRotateCoordinateSystem(protate)
chartVu.SetFractionalZViewportDepth(0.1)
.
.
.
Dim thePlot1 As New FloatingBarPlot(pTransform1)
Dim attrib1 As New ChartAttribute(Color.Black, 1, DashStyle.Solid, Color.Green)
attrib1.SetFillFlag(True)
thePlot1.InitFloatingBarPlot(Dataset1, 0.75, attrib1, ChartObj.JUSTIFY_CENTER)
thePlot1.SetBarOrient(ChartObj.HORIZ_DIR)
chartVu.AddChartObject(thePlot1)

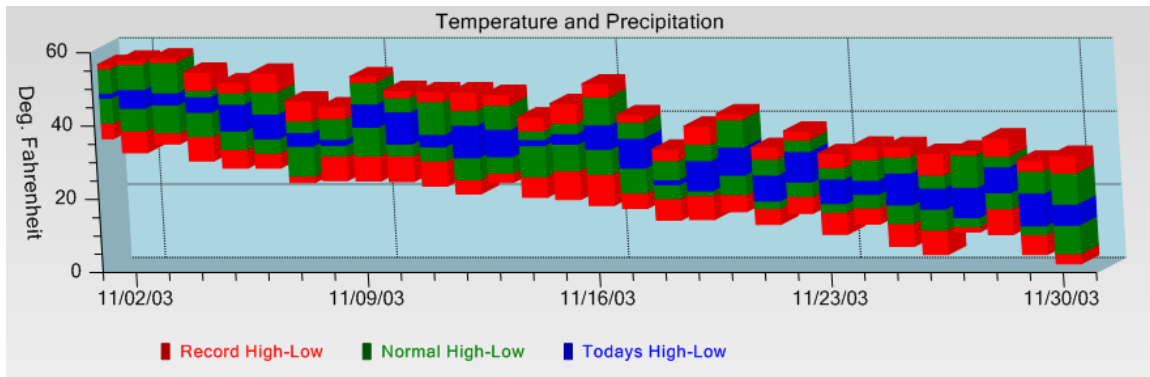
```

Class FloatingStackedBarPlot

GraphObj



The **FloatingStackedBarPlot** class extends the **GroupPlot** class and displays stacked floating bar plots. In a stacked floating bar plot each group is stacked on top of one another, each group bar a cumulative sum of the related group items before it. The bars are free floating because each bar does not reference a fixed base value, as do stacked bars. All bars in a given **FloatingStackedBarPlot** object have the same width. When used with a 2½ D dataset, the position of the each bar with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property specifies the 3D depth of the bar. When a stacked floating bar plot is attached to a 3D dataset, each bar is not restricted to a z-axis plane and can traverse 3D space unrestricted. The individual elements of the bar plot are normally 3D bars, but you can also set the bar plots **BarShapeOption** property to a cylinder (CYLINDER3D).



Floating stacked bar plot (Bargraphs.BostonClimate)

FloatingBarPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal rbarwidth As Double, _
    ByVal attribs As ChartAttribute(), _
    ByVal nbarjust As Integer _
)

[C#]
public FloatingStackedBar(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    double rbarwidth,
    ChartAttribute[] attribs,
    int nbarjust
);
```

<i>transform</i>	The coordinate system for the new FloatingStackedBarPlot object.
<i>dataset</i>	The floating stacked bar graph represents the values in this group dataset.
<i>rbarwidth</i>	The width of the stacked bars in units of the independent axis.
<i>attribs</i>	An array of ChartAttribute objects, sized the same as the number of groups in the dataset, that specify the attributes (outline color and fill color) for each group of a stacked floating bar graph.
<i>nbarjust</i>	The stacked bars are justified with respect to the x-values in the dataset using the <i>rbarjust</i> justification value (JUSTIFY_MIN, JUSTIFY_CENTER, or JUSTIFY_MAX).

**A stacked floating bar plot example from the example program
Bargraphs.BostonClimate)****[C#]**

```

// Create Temperature Chart
// Create a time scale
TimeCoordinates pTransform1 =
    new TimeCoordinates(ChartObj.TIME_SCALE, ChartObj.LINEAR_SCALE );
// Auto-scale time scale based on the the three datasets in the dataset array
pTransform1.AutoScale(HighsAndLows,ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_FAR);

// Set chart rotation
Point3D protate = new Point3D(10, 20, 0);
pTransform1.AbsRotateCoordinateSystem( protate);
chartVu.SetFractionalZViewportDepth(0.1);
.
.
.

// Define attributes of Record High-Low floating bar plot
ChartAttribute RecordHLAttrib =
    new ChartAttribute(Color.Red, 1,DashStyle.Solid, Color.Red);
// Define attributes of Normal High-Low floating bar plot
ChartAttribute NormalHLAttrib =
    new ChartAttribute(Color.Green, 1,DashStyle.Solid, Color.Green);

// Define attributes of Todsyas High-Low floating bar plot
ChartAttribute TodaysHLAttrib =
    new ChartAttribute( Color.Blue, 1,DashStyle.Solid, Color.Blue);

ChartAttribute [] attribs = {RecordHLAttrib, NormalHLAttrib,
    TodaysHLAttrib, NormalHLAttrib, RecordHLAttrib, RecordHLAttrib};
// Create floating bar plot
FloatingStackedBar thePlot1 =
    new FloatingStackedBar(pTransform1, HighsAndLows,

ChartCalendar.GetCalendarWidthValue(ChartObj.DAY_OF_MONTH,0.75),
    attribs, ChartObj.JUSTIFY_CENTER);
chartVu.AddChartObject(thePlot1);

```

[VB]

```

' Create Temperature Chart
' Create a time scale
Dim pTransform1 As New TimeCoordinates(CharObj.TIME_SCALE, CharObj.LINEAR_SCALE)

' Auto-scale time scale based on the the three datasets in the dataset array
pTransform1.AutoScale(HighsAndLows, CharObj.AUTOAXES_NEAR, CharObj.AUTOAXES_FAR)

' Set chart rotation
Dim protate As New Point3D(10, 20, 0)
pTransform1.AbsRotateCoordinateSystem(protate)
chartVu.SetFractionalZViewportDepth(0.1)
.
.
.

' Define attributes of Record High-Low floating bar plot
Dim RecordHLAttrib As New ChartAttribute(Color.Red, 1, DashStyle.Solid, Color.Red)
' Define attributes of Normal High-Low floating bar plot
Dim NormalHLAttrib As New ChartAttribute(Color.Green, 1, DashStyle.Solid,
Color.Green)
' Define attributes of Todsyas High-Low floating bar plot
Dim TodaysHLAttrib As New ChartAttribute(Color.Blue, 1, DashStyle.Solid,
Color.Blue)

Dim attribs As ChartAttribute() = {RecordHLAttrib, NormalHLAttrib, TodaysHLAttrib,
NormalHLAttrib, RecordHLAttrib, RecordHLAttrib}

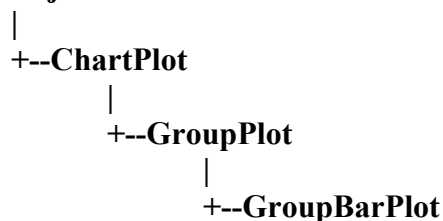
' Create floating bar plot
Dim thePlot1 As New FloatingStackedBar(pTransform1, HighsAndLows,
ChartCalendar.GetCalendarWidthValue(CharObj.DAY_OF_MONTH, 0.75), attribs,
CharObj.JUSTIFY_CENTER)
chartVu.AddChartObject(thePlot1)

```

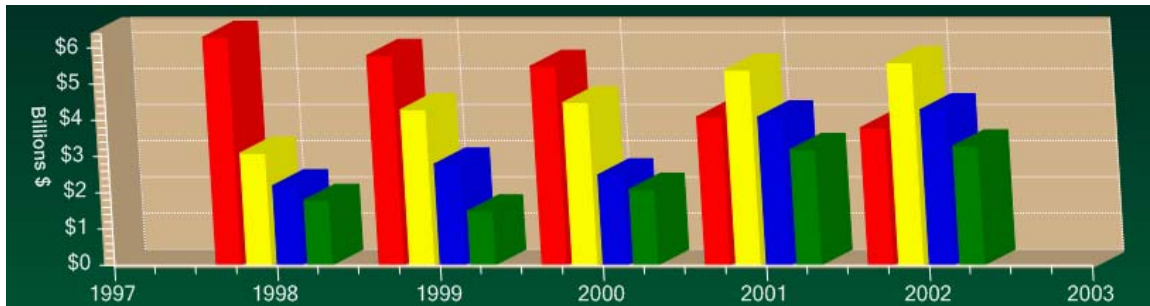
Group Bar Plots

Class GroupBarPlot

GraphObj



The **GroupBarPlot** class extends the **GroupPlot** class and displays data in a group bar format. Individual bars, the height of which corresponds to the group values ($Y[0]$, $Y[1]$, $Y[2]$, ...) of the dataset, are displayed side by side, as a group, justified with respect to the X-position value for each group. When used with a 2½ D dataset, the position of the each bar with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property specifies the 3D depth of the bar. When a group bar plot is attached to a 3D dataset, each bar is not restricted to a z-axis plane and can traverse 3D space unrestricted. The individual elements of the bar plot are normally 3D bars, but you can also set the bar plots **BarShapeOption** property to a cylinder (CYLINDER3D) or a cone (CONE3D).



Group bar plots (Bargraphs.GroupBargraphs)

GroupBarPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal rbarwidth As Double, _
    ByVal rbarbase As Double, _
    ByVal attrs As ChartAttribute(), _
    ByVal nbarjust As Integer _
)

[C#]
public GroupBarPlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    double rbarwidth,
    double rbarbase,
    ChartAttribute[] attrs,
    int nbarjust
);
```

transform

The coordinate system for the new **GroupPlot** object.

<i>dataset</i>	The group bar graph represents the values in this group dataset. Individual bars, the height of which corresponds to the group values (Y[0], Y[1], Y[2], ...) of the dataset.
<i>rbarwidth</i>	The width of the group bars in units of the independent axis. All bars within a group are squeezed into the width defined by <i>rbarwidth</i> . Each individual bar within the group has a width of <i>rbarwidth/dataset.GetNumberGroups()</i> .
<i>rbarbase</i>	The group bars start at the value <i>rbarbase</i> , and extend to the group bar values represented by the dataset.
<i>attribs</i>	An array of ChartAttribute objects, sized the same as the number of groups in the dataset specify the attributes (outline color and fill color) for each group of a group bar graph.
<i>nbarjust</i>	The group bars are justified with respect to the x-values in the dataset using the <i>rbarjust</i> justification value (JUSTIFY_MIN, JUSTIFY_CENTER, or JUSTIFY_MAX).

The attributes for each group can set or modified using the SetSegment... methods, where the segment number parameter cooresponds to the group number. These methods include **SetSegmentAttributes**, **SetSegmentFillColor**, **SetSegmentLineColor**, and **SetSegmentColor**.

Group bar plot example (extracted from the example program Bargraphs.GroupBargraphs)

[C#]

```
// Define data values
int nNumPnts = 5, nNumGroups = 4;
ChartCalendar []xValues= new ChartCalendar[nNumPnts];
double [,]groupBarData = new double[nNumGroups,nNumPnts];
theFont = new Font("Microsoft Sans Serif", 10, FontStyle.Bold);
xValues[0] = new ChartCalendar(1998, ChartObj.JANUARY, 1);
groupBarData[0,0] = 6.3; groupBarData[1,0] = 3.1;
groupBarData[2,0] = 2.2; groupBarData[3,0] = 1.8;

xValues[1] = new ChartCalendar(1999, ChartObj.JANUARY, 1);
groupBarData[0,1] = 5.8; groupBarData[1,1] = 4.3;
groupBarData[2,1] = 2.8; groupBarData[3,1] = 1.5;
.
.
```

278 *Group Plot Objects*

```
.  
// define dataset  
TimeGroupDataset Dataset1 = new  
TimeGroupDataset("GroupTimeData",xValues,groupBarData);  
Dataset1.ImplicitDepthValue = 1.0;  
  
// DEFINE FIRST GRAPH AS A GROUP BAR GRAPH  
// define coordinate system of first chart  
pTransform1 = new TimeCoordinates();  
pTransform1.AutoScale(Dataset1,ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR);  
  
// force x-axis time scale to a specific range  
pTransform1.SetTimeScaleStart(new ChartCalendar(1997,ChartObj.JANUARY,1));  
pTransform1.SetTimeScaleStop(new ChartCalendar(2003,ChartObj.JANUARY,1));  
// force starting position of y-scale to 0  
pTransform1.SetScaleStartY(0);  
  
// define position of chart in graph area  
pTransform1.SetGraphBorderDiagonal(0.1, .15, .9, 0.45) ;  
  
// define chart rotation  
Point3D protate = new Point3D(10, 20, 0);  
pTransform1.AbsRotateCoordinateSystem( protate);  
chartVu.SetFractionalZViewportDepth(0.1);  
  
.br/>.br/>.br/>  
// define color attributes the group plot  
ChartAttribute attrib1 = new ChartAttribute(Color.Red, 1,DashStyle.Solid,  
Color.Red);  
ChartAttribute attrib2 = new ChartAttribute(Color.Yellow, 1,DashStyle.Solid,  
Color.Yellow);  
ChartAttribute attrib3 = new ChartAttribute(Color.Blue, 1,DashStyle.Solid,  
Color.Blue);  
ChartAttribute attrib4 = new ChartAttribute(Color.Green, 1,DashStyle.Solid,  
Color.Green);  
ChartAttribute []attribArray = {attrib1, attrib2, attrib3, attrib4};  
// create a group bar plot  
GroupBarPlot thePlot1 = new GroupBarPlot(pTransform1, Dataset1,  
ChartCalendar.GetCalendarWidthValue(ChartObj.YEAR,0.75), 0.0, attribArray,  
ChartObj.JUSTIFY_CENTER);  
thePlot1.SetBarOverlap(0.0);  
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```

' Define data values
Dim nNumPnts As Integer = 5
Dim nNumGroups As Integer = 4
Dim xValues(nNumPnts - 1) As ChartCalendar
Dim groupBarData(nNumGroups - 1, nNumPnts - 1) As Double
theFont = New Font("Microsoft Sans Serif", 10, FontStyle.Bold)
xValues(0) = New ChartCalendar(1998, ChartObj.JANUARY, 1)
groupBarData(0, 0) = 6.3
groupBarData(1, 0) = 3.1
groupBarData(2, 0) = 2.2
groupBarData(3, 0) = 1.8

xValues(1) = New ChartCalendar(1999, ChartObj.JANUARY, 1)
groupBarData(0, 1) = 5.8
groupBarData(1, 1) = 4.3
groupBarData(2, 1) = 2.8
groupBarData(3, 1) = 1.5

.
.
.

' define dataset
Dim Dataset1 As New TimeGroupDataset("GroupTimeData", xValues, groupBarData)
Dataset1.ImplicitDepthValue = 1.0

' DEFINE FIRST GRAPH AS A GROUP BAR GRAPH
' define coordinate system of first chart
Dim pTransform1 As New TimeCoordinates()
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR)

' force x-axis time scale to a specific range
pTransform1.SetTimeScaleStart(New ChartCalendar(1997, ChartObj.JANUARY, 1))
pTransform1.SetTimeScaleStop(New ChartCalendar(2003, ChartObj.JANUARY, 1))
' force starting position of y-scale to 0
pTransform1.SetScaleStartY(0)

' define position of chart in graph area
pTransform1.SetGraphBorderDiagonal(0.1, 0.15, 0.9, 0.45)

' define chart rotation

```

```

Dim protate As New Point3D(10, 20, 0)
pTransform1.AbsRotateCoordinateSystem(protate)
chartVu.SetFractionalZViewportDepth(0.1)

.
.
.

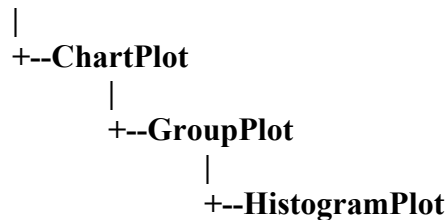
' define color attributes the group plot
Dim attrib1 As New ChartAttribute(Color.Red, 1, DashStyle.Solid, Color.Red)
Dim attrib2 As New ChartAttribute(Color.Yellow, 1, DashStyle.Solid, Color.Yellow)
Dim attrib3 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid, Color.Blue)
Dim attrib4 As New ChartAttribute(Color.Green, 1, DashStyle.Solid, Color.Green)
Dim attribArray As ChartAttribute() = {attrib1, attrib2, attrib3, attrib4}
' create a group bar plot
Dim thePlot1 As New GroupBarPlot(pTransform1, Dataset1, _
    ChartCalendar.GetCalendarWidthValue(ChartObj.YEAR, 0.75), _
    0.0, attribArray, ChartObj.JUSTIFY_CENTER)
thePlot1.SetBarOverlap(0.0)
chartVu.AddChartObject(thePlot1)

```

Histogram Plots

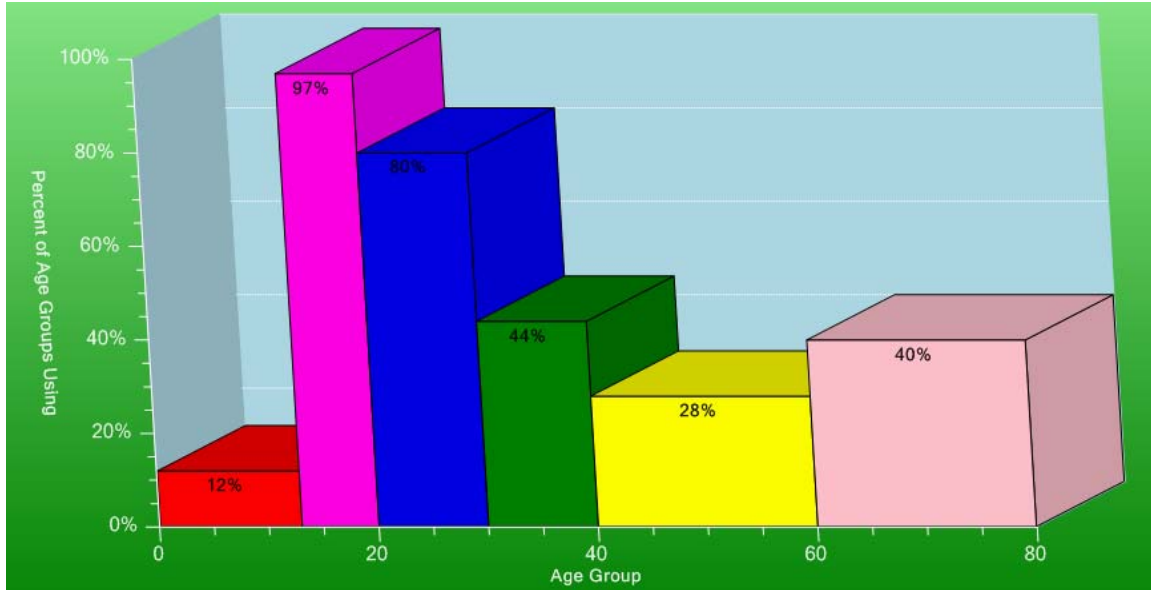
Class HistogramPlot

GraphObj



The **HistogramPlot** class extends the **GroupPlot** class and displays histogram plots. A histogram plot is a collection of rectangular objects with independent positions, widths and heights, specified using the values of the associated group dataset. The number of groups must be two. The X-values of the group dataset represent the x-position of the lower left corner of each histogram bar, the Y[0] values set the height of each histogram bar, and the Y[1] values set the width of each histogram bar. The histogram bars share a common base value. When used with a 2½ D dataset, the position of the each bar with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property specifies the 3D depth of the bar. When a

histogram bar plot is attached to a 3D dataset, each bar is not restricted to a z-axis plane and can traverse 3D space unrestricted. The individual elements of the histogram plot are normally 3D bars, but you can also set the bar plots **BarShapeOption** property to a cylinder (CYLINDER3D) or a cone (CONE3D).



Histogram plots (Bargraphs.HistogramBars)

Histogram constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal rbarbase As Double, _
    ByVal attrib As ChartAttribute _
)

[C#]
public HistogramPlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    double rbarbase,
    ChartAttribute attrib
);
```

transform

The coordinate system for the new **HistogramPlot** object.

dataset

The histogram plot represents the values in this group dataset. The number of groups must be two. The X-values of the group dataset represent the x-position of the lower left corner of each histogram bar, the Y[0] values set the height of each histogram bar, and the Y[1] values set the width of each histogram bar.

<i>rbarbase</i>	The histogram bars start at the value <i>rbarbase</i> , and extend to the histogram bar values represented by the dataset.
<i>attrib</i>	Specifies the attributes (line color and line style) for the histogram bars.

An individual histogram bar in the histogram plot object can have unique attributes. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods in the manner described for **SimplePlot** objects. Each histogram bar can be labeled with the Y[0] group value bar (bar height) using the bar data point methods, see the example below.

Histogram plot example (extracted from the example program **Bargraphs.HistogramBars**)

[C#]

```
// Define data
int numpnts = 6;
int numgroups = 2;
double []x1= new double[numpnts];
double [,]y1 = new double[numgroups,numpnts];
//           height           width
x1[0] = 0;  y1[0,0] = .12; y1[1,0] = 13;
x1[1] = 13; y1[0,1] = .97; y1[1,1] = 7;
x1[2] = 20; y1[0,2] = .80; y1[1,2] = 10;
x1[3] = 30; y1[0,3] = .44; y1[1,3] = 10;
x1[4] = 40; y1[0,4] = .28; y1[1,4] = 20;
x1[5] = 60; y1[0,5] = .4; y1[1,5] = 20;

// Define dataset
GroupDataset Dataset1 = new GroupDataset("Actual Sales",x1,y1);
Dataset1.ImplicitDepthValue = 1.0;
Dataset1.ImplicitMinZValue = 0.0;

// Define coordinate system
CartesianCoordinates pTransform1 = new CartesianCoordinates();
pTransform1.SetScaleStartY(0);
pTransform1.SetScaleStartX(0);
pTransform1.SetScaleStopX(80);
```

```

pTransform1.SetScaleStopY(1.00);

// position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, .15, .9, 0.825) ;
// Set chart rotation
Point3D protate = new Point3D(10, 20, 0);
pTransform1.AbsRotateCoordinateSystem( protate);
.
.
.
// define histogram plot
ChartAttribute attrib1 = new ChartAttribute (Color.Black, 0,DashStyle.Solid,
Color.Green);
attrib1.SetFillFlag(true);
HistogramPlot thePlot1 = new HistogramPlot(pTransform1, Dataset1, 0.0, attrib1);
NumericLabel bardatavalue = thePlot1.GetPlotLabelTemplate();
// define data point values for histogram plot
bardatavalue.SetTextFont(theFont);
bardatavalue.SetNumericFormat (ChartObj.PERCENTFORMAT);
bardatavalue.SetColor(Color.Black);
thePlot1.SetBarDatapointLabelPosition(ChartObj.INSIDE_BAR);
thePlot1.SetPlotLabelTemplate(bardatavalue);
thePlot1.SetShowDatapointValue(true);

// color each histogram bar different
thePlot1.SetSegmentAttributesMode(true);
thePlot1.SetSegmentFillColor(0,Color.Red);
thePlot1.SetSegmentFillColor(1, Color.Magenta);
thePlot1.SetSegmentFillColor(2, Color.Blue);
thePlot1.SetSegmentFillColor(3, Color.Green);
thePlot1.SetSegmentFillColor(4, Color.Yellow);
thePlot1.SetSegmentFillColor(5, Color.Pink);
chartVu.AddChartObject(thePlot1);

```

[Visual Basic]

```

' Define data
Dim numpnts As Integer = 6
Dim numgroups As Integer = 2
Dim x1(numpnts - 1) As Double
Dim y1(numgroups - 1, numpnts - 1) As Double
'      height      width
x1(0) = 0

```

284 *Group Plot Objects*

```
y1(0, 0) = 0.12
y1(1, 0) = 13
x1(1) = 13
y1(0, 1) = 0.97
y1(1, 1) = 7
x1(2) = 20
y1(0, 2) = 0.8
y1(1, 2) = 10
x1(3) = 30
y1(0, 3) = 0.44
y1(1, 3) = 10
x1(4) = 40
y1(0, 4) = 0.28
y1(1, 4) = 20
x1(5) = 60
y1(0, 5) = 0.4
y1(1, 5) = 20

' Define dataset
Dim Dataset1 As New GroupDataset("Actual Sales", x1, y1)
Dataset1.ImplicitDepthValue = 1.0
Dataset1.ImplicitMinZValue = 0.0

' Define coordinate system
Dim pTransform1 As New CartesianCoordinates()
pTransform1.SetScaleStartY(0)
pTransform1.SetScaleStartX(0)
pTransform1.SetScaleStopX(80)
pTransform1.SetScaleStopY(1.0)

' position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 0.825)

' Set chart rotation
Dim protate As New Point3D(10, 20, 0)
pTransform1.AbsRotateCoordinateSystem(protate)

.
.
.

' define histogram plot
Dim attrib1 As New ChartAttribute(Color.Black, 0, DashStyle.Solid, Color.Green)
attrib1.SetFillFlag(True)
Dim thePlot1 As New HistogramPlot(pTransform1, Dataset1, 0.0, attrib1)
```



```

Dim bardatavalue As NumericLabel = thePlot1.GetPlotLabelTemplate()
' define data point values for histogram plot
bardatavalue.SetTextFont(theFont)
bardatavalue.SetNumericFormat(ChartObj.PERCENTFORMAT)
bardatavalue.SetColor(Color.Black)
thePlot1.SetBarDatapointLabelPosition(ChartObj.INSIDE_BAR)
thePlot1.SetPlotLabelTemplate(bardatavalue)
thePlot1.SetShowDatapointValue(True)

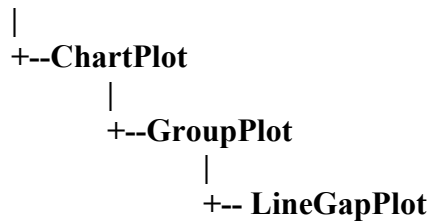
' color each histogram bar different
thePlot1.SetSegmentAttributesMode(True)
thePlot1.SetSegmentFillColor(0, Color.Red)
thePlot1.SetSegmentFillColor(1, Color.Magenta)
thePlot1.SetSegmentFillColor(2, Color.Blue)
thePlot1.SetSegmentFillColor(3, Color.Green)
thePlot1.SetSegmentFillColor(4, Color.Yellow)
thePlot1.SetSegmentFillColor(5, Color.Pink)
chartVu.AddChartObject(thePlot1)

```

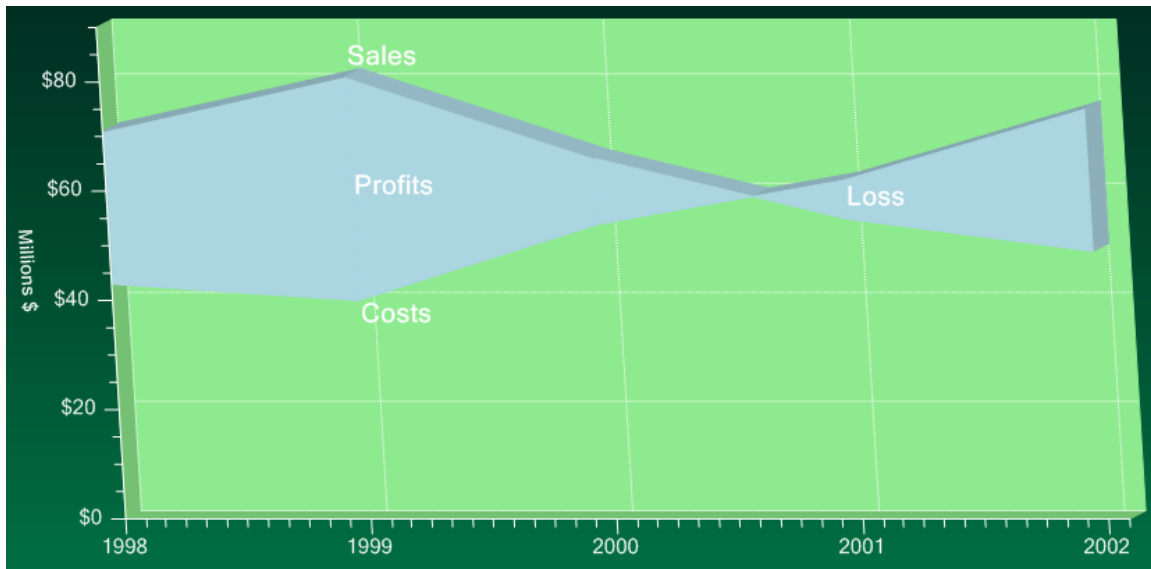
Line Gap Plots

Class LineGapPlot

GraphObj



The **LineGapPlot** class extends the **GroupPlot** class and displays a line gap chart. The number of groups must be two. A line gap chart consists of two line plots where a contrasting color fills and highlights the area between the two lines. The (X, Y[0]) values of the group dataset represent the first of the bounding lines, and the (X, Y[1]) values of the group dataset represent the second of the bounding lines. This plot type only works with a 2½ D dataset. The position of the line gap plot with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property specifies the 3D depth of the line gap plot.



Line gap plot (MiscCharts.LineGap)

LineGapPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal attrib As ChartAttribute _
)

[C#]
public LineGapPlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    ChartAttribute attrib
);
```

<i>transform</i>	The coordinate system for the new LineGapPlot object.
<i>dataset</i>	The line gap plot represents the values in this group dataset. The number of groups in this group dataset must be two.
<i>attrib</i>	Specifies the attributes (line and fill color) for the fill area.

A segment between adjacent x-values in the line gap plot object can have unique attributes. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods in the manner described for **SimplePlot** objects.

Line gap bar plot example (extracted from the example program MiscCharts.LineGap)

[C#]

```
// Simulate data
int nNumPnts = 5, nNumGroups = 2;
ChartCalendar []xValues= new ChartCalendar[nNumPnts];
double [,]groupBarData = new double[nNumGroups,nNumPnts];

xValues[0] = new ChartCalendar(1998, ChartObj.JANUARY, 1);
groupBarData[0,0] = 43; groupBarData[1,0] = 71;

xValues[1] = new ChartCalendar(1999, ChartObj.JANUARY, 1);
groupBarData[0,1] = 40; groupBarData[1,1] = 81;

xValues[2] = new ChartCalendar(2000, ChartObj.JANUARY, 1);
groupBarData[0,2] = 54; groupBarData[1,2] = 66;

xValues[3] = new ChartCalendar(2001, ChartObj.JANUARY, 1);
groupBarData[0,3] = 62; groupBarData[1,3] = 55;

xValues[4] = new ChartCalendar(2002, ChartObj.JANUARY, 1);
groupBarData[0,4] = 75; groupBarData[1,4] = 49;

// Create a group dataset
TimeGroupDataset Dataset1 = new
TimeGroupDataset("GroupTimeData",xValues,groupBarData);
Dataset1.ImplicitDepthValue = 1.0;

// Define a coordinate system
pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(Dataset1,ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_FAR);
.
.
.
.

// Define the line gap plot
ChartAttribute attrib1 = new ChartAttribute(Color.LightBlue, 1,DashStyle.Solid,
Color.LightBlue);
attrib1.SetFillFlag(true);
attrib1.SetLineFlag(true);
LineGapPlot thePlot1 = new LineGapPlot(pTransform1, Dataset1, attrib1);
```

288 *Group Plot Objects*

```
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
Dim nNumPnts As Integer = 5
Dim nNumGroups As Integer = 2
Dim xValues(nNumPnts - 1) As ChartCalendar
Dim groupBarData(nNumGroups - 1, nNumPnts - 1) As Double

theFont = New Font("Microsoft Sans Serif", 10, FontStyle.Bold)
xValues(0) = New ChartCalendar(1998, ChartObj.JANUARY, 1)
groupBarData(0, 0) = 43
groupBarData(1, 0) = 71

xValues(1) = New ChartCalendar(1999, ChartObj.JANUARY, 1)
groupBarData(0, 1) = 40
groupBarData(1, 1) = 81

xValues(2) = New ChartCalendar(2000, ChartObj.JANUARY, 1)
groupBarData(0, 2) = 54
groupBarData(1, 2) = 48

xValues(3) = New ChartCalendar(2001, ChartObj.JANUARY, 1)
groupBarData(0, 3) = 56
groupBarData(1, 3) = 44

xValues(4) = New ChartCalendar(2002, ChartObj.JANUARY, 1)
groupBarData(0, 4) = 58
groupBarData(1, 4) = 40
Dim Dataset1 As New TimeGroupDataset("GroupTimeData", xValues, groupBarData)

Dim pTransform1 As New TimeCoordinates()
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_FAR)

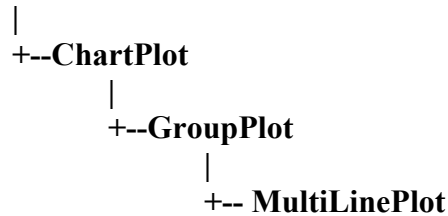
.
.
.

Dim attrib1 As New ChartAttribute(Color.Black, 1, DashStyle.Solid, Color.Red)
attrib1.SetFillFlag(True)
attrib1.SetLineFlag(False)
Dim thePlot1 As New LineGapPlot(pTransform1, Dataset1, attrib1)
chartVu.AddChartObject(thePlot1)
```

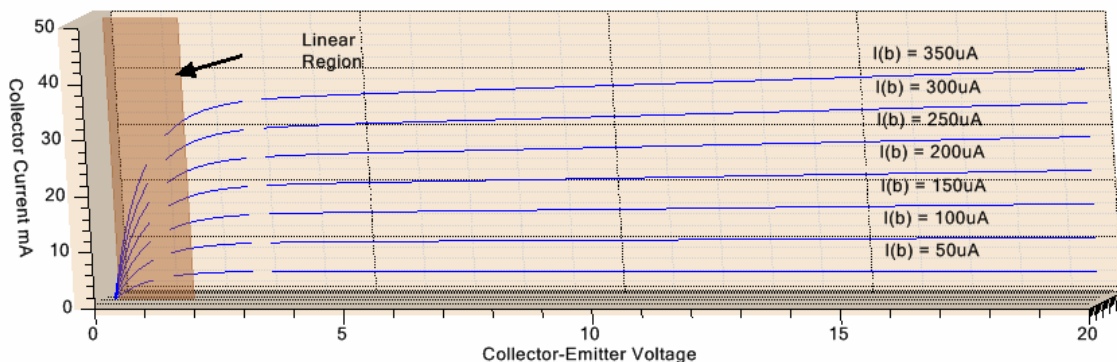
Multi-Line Plots

Class MultiLinePlot

GraphObj



The **MultiLinePlot** class extends the **GroupPlot** class and displays group data in multi-line format. A group dataset with eight groups will display eight separate line plots. The y-values for each group of the dataset are the y-values for each line in the plot. Each line plot share the same x-values of the group dataset. When used with a 2½ D dataset, the position of all of the lines with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** is ignored since a simple line plot has no depth.



Multi-line plot (same as MultilinePlots.MultiLines1)

MultiLinePlot constructor

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal attribs As ChartAttribute() _
)

[C#]
public MultiLinePlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
  
```

```
ChartAttribute[] attribs
);
```

<i>transform</i>	The coordinate system for the new MultiLinePlot object.
<i>dataset</i>	The multi-line plot represents the values in this group dataset.
<i>attribs</i>	An array of ChartAttribute objects, sized the same as the number of groups in the dataset, to specify the attributes (line color and line style) for each group of the multi-line plot.

The attributes for each group can be set or modified using the **SetSegment...** methods, where the segment number parameter corresponds to the group number. These methods include **SetSegmentAttributes**, **SetSegmentFillColor**, **SetSegmentLineColor**, and **SetSegmentColor**.

Multi-line plot example (extracted from the example program **MultiLinePlots.MultiLine1**)

[C#]

```
// Simulate data
int  numPoints = 100;
int  numGroups = 7;
double []x1 = new double[numPoints];
double [,]y1 = new double[numGroups,numPoints];
int i, j;

for (i=0; i < numPoints; i++)
{
    x1[i] = (double)i * 0.2;
    for (j = 0; j < numGroups; j++)
        y1[j,i] = j * (i * 0.01) + (double)(j+1) * 5.0 * (1.0 - Math.Exp(-x1[i]/0.7))
;
}

// Initialize a couple of data values with the bad data value constant
y1[0,5] = ChartObj.rBadDataValue;
y1[3,15] = ChartObj.rBadDataValue;

// Create a group dataset to hold all of the multi-line data
```

```

GroupDataset Dataset1 = new GroupDataset("First",x1,y1);
    Dataset1.ImplicitDepthValue = 0.2;
    Dataset1.ImplicitZValue = 0.6;

// Auto-scale a coordinate system
pTransform1 = new CartesianCoordinates( ChartObj.LINEAR_SCALE,
ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
// Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, .15, .90, 0.75) ;

// Force starting values of x- and y-scale to 0
pTransform1.SetScaleStartX(0);
pTransform1.SetScaleStartY(0);

// Set rotate parameters
pTransform1.AbsRotateCoordinateSystem(new Point3D(10,20,0));
chartVu.SetFractionalZViewportDepth(0.1);

.
.
.

// Define attributes for multi-line plot
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid);
ChartAttribute []attribArray = new ChartAttribute[numGroups];
for (i=0; i < numGroups; i++)
    attribArray[i] = (ChartAttribute) attrib1.Clone();

// Define a multi-line plot
MultiRibbonPlot thePlot1 = new MultiRibbonPlot(pTransform1, Dataset1,
attribArray);
chartVu.AddChartObject(thePlot1);

```

[Visual Basic]

```

' Simulate data
Dim numPoints As Integer = 100
Dim numGroups As Integer = 7
Dim x1(numPoints - 1) As Double
Dim y1(numGroups - 1, numPoints - 1) As Double
Dim i, j As Integer

```

292 *Group Plot Objects*

```
For i = 0 To numPoints - 1
    x1(i) = CDBl(i) * 0.2
    For j = 0 To numGroups - 1
        y1(j, i) = j * (i * 0.01) + CDBl(j + 1) * 5.0 * (1.0 - Math.Exp((-x1(i) / 0.7)))
    Next j
Next i ' Initialize a couple of data values with the bad data value constant
y1(0, 5) = ChartObj.rBadDataValue
y1(3, 15) = ChartObj.rBadDataValue

' Create a group dataset to hold all of the multi-line data
Dim Dataset1 As New GroupDataset("First", x1, y1)
Dataset1.ImplicitDepthValue = 0.2
Dataset1.ImplicitZValue = 0.6

' Auto-scale a coordinate system
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE,
ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
' Position chart in graph area
pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 0.75)

' Force starting values of x- and y-scale to 0
pTransform1.SetScaleStartX(0)
pTransform1.SetScaleStartY(0)

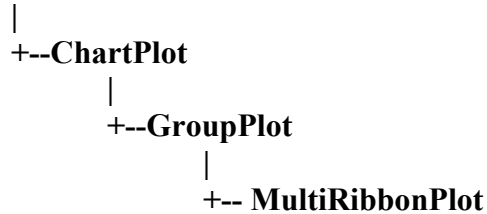
' Set rotate parameters
pTransform1.AbsRotateCoordinateSystem(New Point3D(20, 30, 0))
chartVu.SetFractionalZViewportDepth(0.1)

.
.
.

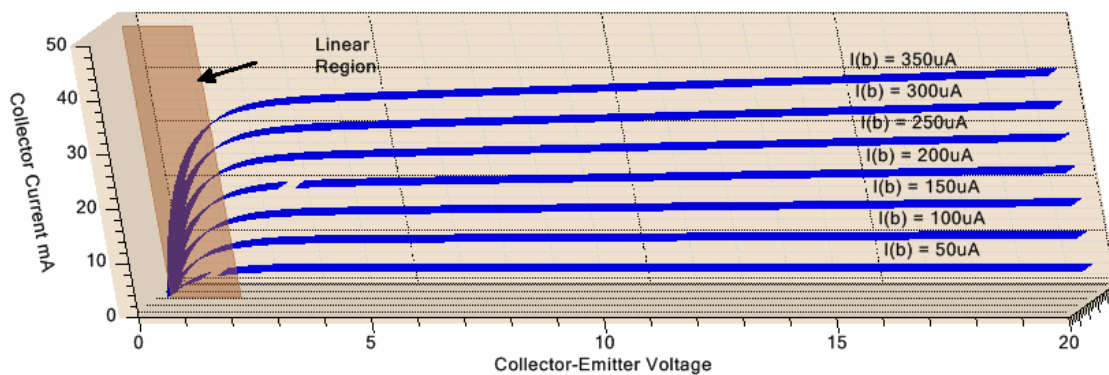
' Define attributes for multi-line plot
Dim attrib1 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid)
Dim attribArray(numGroups) As ChartAttribute
For i = 0 To numGroups - 1
    attribArray(i) = CType(attrib1.Clone(), ChartAttribute)
Next i
' Define a multi-line plot
Dim thePlot1 As New MultiLinePlot(pTransform1, Dataset1, attribArray);
chartVu.AddChartObject(thePlot1)
```


Class MultiRibbonPlot

GraphObj



The **MultiRibbonPlot** class extends the **GroupPlot** class and displays group data in multi-ribbon plot format. A group dataset with eight groups will display eight separate multi-ribbon plots. The y-values for each group of the dataset are the y-values for each line in the plot. Each line plot share the same x-values of the group dataset. This group plot only works with a 2½ D dataset. The position of all of the ribbons of the multi-ribbon with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** sets the 3D depth for all of the ribbons.



Multi-ribbon plots (MultiLinePlots.MultiLines1)

MultiRibbonPlot constructor

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal attrs As ChartAttribute() _
)

[C#]
public MultiRibbonPlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    ChartAttribute[] attrs
);
  
```

<i>transform</i>	The coordinate system for the new MultiLinePlot object.
<i>dataset</i>	The multi-line plot represents the values in this group dataset.
<i>attribs</i>	An array of ChartAttribute objects, sized the same as the number of groups in the dataset, to specify the attributes (line color and line style) for each group of the multi-line plot.

The attributes for each group can be set or modified using the **SetSegment...** methods, where the segment number parameter corresponds to the group number. These methods include **SetSegmentAttributes**, **SetSegmentFillColor**, **SetSegmentLineColor**, and **SetSegmentColor**.

Multi-ribbon plot example (extracted from the example program **MultiLinePlots.MultiLine**)

[C#]

```
.
.
.
// Define attributes for multi-line plot
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1, DashStyle.Solid);
ChartAttribute []attribArray = new ChartAttribute[numGroups];
for (i=0; i < numGroups; i++)
    attribArray[i] = (ChartAttribute) attrib1.Clone();

// Define a multi-line plot
MultiRibbonPlot thePlot1 = new MultiRibbonPlot(pTransform1, Dataset1,
attribArray);
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
.
.
.
' Define attributes for multi-line plot
Dim attrib1 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid)
```

```

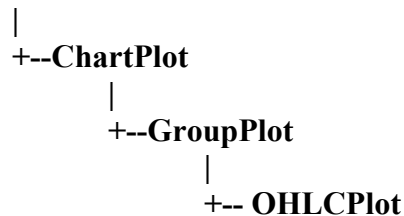
Dim attribArray(numGroups) As ChartAttribute
For i = 0 To numGroups - 1
    attribArray(i) = CType(attrib1.Clone(), ChartAttribute)
Next i
' Define a multi-line plot
Dim thePlot1 As New MultiRibbonPlot(pTransform1, Dataset1, attribArray)
' MultiLinePlot thePlot1 = new MultiLinePlot(pTransform1, Dataset1,
attribArray);
chartVu.AddChartObject(thePlot1)

```

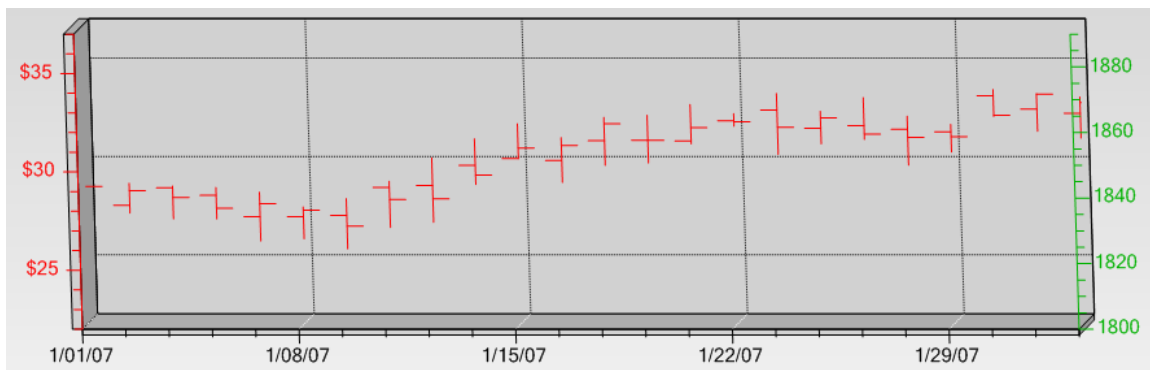
Open-High-Low-Close Plots

Class OHLCPlot

GraphObj



The **OHLCPlot** class extends the **GroupPlot** class and displays stock market data in an open-high-low-close format common in financial technical analysis. Every item of the plot is a vertical line, representing High and Low values, with two small horizontal "flags", one left and one right extending from the vertical High-Low line and representing the Open and Close values. The number of groups must be four. The Y[0] values of the group dataset represent the values for Open, the Y[1] values for High, the Y[2] values for Low, and the Y[3] values for Close. When used with a 2½ D dataset, the position of the each OHLC grouping with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** is ignored. When a OHLC plot is attached to a 3D dataset, each OHLC group is not restricted to a z-axis plane and can traverse 3D space unrestricted.



*Open-High-Low-Close (OHLC) plot (FinancialExamples.OHLCChart)***OHLCPlot constructor**

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal rflagwidth As Double, _
    ByVal attrib As ChartAttribute _
)

[C#]
public OHLCPlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    double rflagwidth,
    ChartAttribute attrib
);

```

<i>transform</i>	The coordinate system for the new OHLCPlot object.
<i>dataset</i>	The OHLCPlot plot will represent the group open-high-low-close values in this group dataset. The number of groups must be four. The Y[0] values of the group dataset represent the values for Open, the Y[1] values for High, the Y[2] values for Low, and the Y[3] values for Close.
<i>rflagwidth</i>	The width of the open and close markers in units of the independent axis.
<i>attrib</i>	Specifies the attributes (line color and line style) for the open-high-low-close plot.

An individual OHLC element in an OHLC plot object can have unique attributes. Use the objects **SetSegmentAttributesMode** and **SetSegmentAttributes** methods in the manner described for **SimplePlot** objects.

**OHLC plot example (extracted from the example program
FinancialExamples.OHLCChart)**

```

[C#]

// CHART #1 - top of chart
// create dataset for ohlc data
double zvalue = 0.5;

```

```

double depthvalue = 0.2;

TimeGroupDataset Dataset1 = new TimeGroupDataset("Stock
Data",xValues,stockPriceData, zvalue, depthvalue);

// define coordinate system for stock data
pTransform1 = new TimeCoordinates();
pTransform1.SetWeekType(weekmode);
pTransform1.AutoScale(Dataset1,ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR);
// position chart at top of graph area
pTransform1.SetGraphBorderDiagonal(0.1, .15, .90, 0.55) ;

// Set rotational parameters
Point3D protate = new Point3D(10, 10, 0);
pTransform1.AbsRotateCoordinateSystem( protate);
// Set chart depth
chartVu.SetFractionalZViewportDepth(0.1);
.
.
.
OHLCPLOT thePlot1 = new OHLCPLOT(pTransform1, Dataset1,
ChartCalendar.GetCalendarWidthValue(ChartObj.DAY_OF_YEAR,0.75),
attrib1);
thePlot1.SetFastClipMode(ChartObj.FASTCLIP_X);
chartVu.AddChartObject(thePlot1);

```

[Visual Basic]

```

' CHART #1 - top of chart
' create dataset for ohlc data
Dim zvalue As Double = 0.5
Dim depthvalue As Double = 0.2
Dim Dataset1 As New TimeGroupDataset("Stock Data", xValues, stockPriceData,
zvalue, depthvalue)

' define coordinate system for stock data
pTransform1 = New TimeCoordinates()
pTransform1.SetWeekType(weekmode)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR)
' position chart at top of graph area
pTransform1.SetGraphBorderDiagonal(0.1, 0.15, 0.9, 0.55)

' Set rotational parameters
Dim protate As New Point3D(10, 10, 0)
pTransform1.AbsRotateCoordinateSystem(protate)

```

```

' Set chart depth
chartVu.SetFractionalZViewportDepth(0.1)

.
.
.

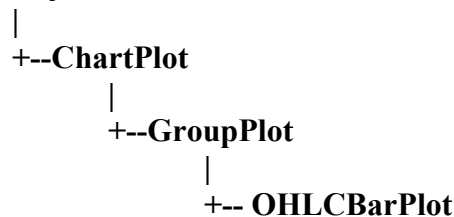
' Define attributes for OHLC plot
Dim attrib1 As New ChartAttribute(Color.Red, 1, DashStyle.Solid, Color.Red)
Dim openHighAttrib As New ChartAttribute(Color.Red, 1, DashStyle.Solid, _
    Color.LimeGreen)
Dim closeHighAttrib As New ChartAttribute(Color.Red, 1, _
    DashStyle.Solid, Color.Yellow)
attrib1.SetFillFlag(True)
' Define OHLC plot
OHLCPlot thePlot1 = new OHLCPlot(pTransform1, Dataset1, _
    ChartCalendar.GetCalendarWidthValue(ChartObj.DAY_OF_YEAR,0.75), _
    attrib1);
thePlot1.SetFastClipMode(ChartObj.FASTCLIP_X)
chartVu.AddChartObject(thePlot1)

```

* Note how the **ChartCalendar.GetCalendarWidthValue** method calculates the width of the bars as a function of time, in this case a width of 0.75 days.

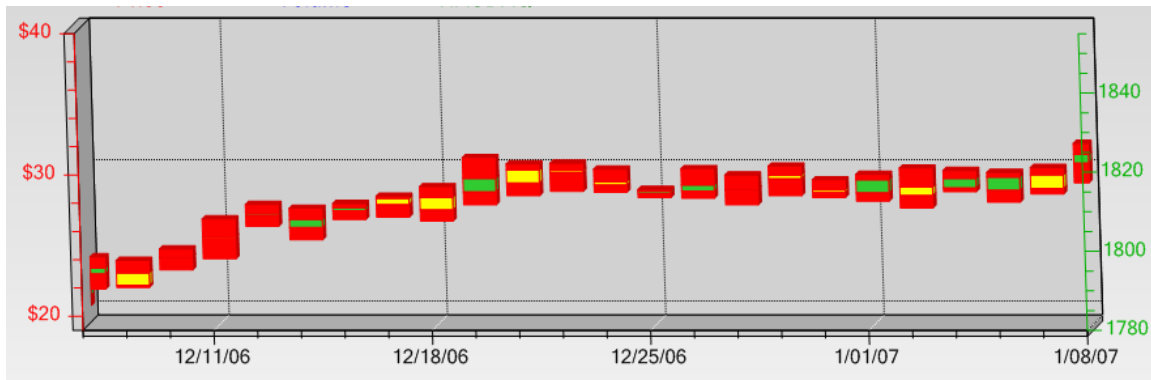
Class OHLCBarPlot

GraphObj



The **OHLCBarPlot** class extends the **GroupPlot** class and displays stock market data in an multi-colored bar format that is easier to view in 3D than the traditional OHLC plot. The endpoints of each bar represent the High and Low values, while the internal colored bar represents the Open and Close values. Different colors are used if the open is higher than the close, or the close is higher than the open. The number of groups must be four. The Y[0] values of the group dataset represent the values for Open, the Y[1] values for High, the Y[2] values for Low, and the Y[3] values for Close. When used with a 2½ D dataset, the position of the each OHLC grouping with respect to the z-axis is set using the

attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** is sets the bar depth. When a OHLC plot is attached to a 3D dataset, each OHLC group is not restricted to a z-axis plane and can traverse 3D space unrestricted.



Ooen-High-Low-Close (OHLC) bar plot (FinancialExamples.OHLCChart)

OHLCBarPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal rflagwidth As Double, _
    ByVal attrib As ChartAttribute _

    ByVal openhighattrib As ChartAttribute, _
    ByVal closehighattrib As ChartAttribute
)

[C#]
public OHLCBarPlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    double rflagwidth,
    ChartAttribute attrib,
    ChartAttribute openhighattrib,
    ChartAttribute closehighattrib
);
```

transform The coordinate system for the new **OHLCPlot** object.

dataset The **OHLCPlot** plot will represent the group open-high-low-close values in this group dataset. The number of groups must be four. The Y[0] values of the group dataset represent the values for Open, the Y[1] values for High, the Y[2] values for Low, and the Y[3] values for Close.

<i>rflagwidth</i>	The width of the open and close markers in units of the independent axis.
<i>attrib</i>	Specifies the main attributes (color) for the open-high-low-close plot.
<i>openhighattrib</i>	Specifies the attribute of the internal bar used to display the open and close values, when the open is higher than the close
<i>closehighattrib</i>	Specifies the attribute of the internal bar used to display the open and close values, when the close is higher than the open.

OHLC plot example (extracted from the example program FinancialExamples.OHLCChart)

[C#]

```
// Define attributes for OHLC bar plot
ChartAttribute attrib1 = new ChartAttribute(Color.Red, 1,
    DashStyle.Solid, Color.Red);
ChartAttribute openHighAttrib = new ChartAttribute(Color.Red, 1,
    DashStyle.Solid, Color.LimeGreen);
ChartAttribute closeHighAttrib = new ChartAttribute(Color.Red, 1,
    DashStyle.Solid, Color.Yellow);
attrib1.SetFillFlag(true);
// Define OHLC plot
OHLCBarPlot thePlot1 = new OHLCBarPlot(pTransform1, Dataset1,
    ChartCalendar.GetCalendarWidthValue(ChartObj.DAY_OF_YEAR, 0.75),
    attrib1, openHighAttrib, closeHighAttrib);
thePlot1.SetFastClipMode(ChartObj.FASTCLIP_X);
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
' Define attributes for OHLC plot
Dim attrib1 As New ChartAttribute(Color.Red, 1, DashStyle.Solid, Color.Red)
Dim openHighAttrib As New ChartAttribute(Color.Red, 1, _
    DashStyle.Solid, Color.LimeGreen)
Dim closeHighAttrib As New ChartAttribute(Color.Red, 1, _
    DashStyle.Solid, Color.Yellow)
attrib1.SetFillFlag(True)
' Define OHLC plot
```



```

Dim thePlot1 As New OHLCBarPlot(pTransform1, Dataset1, _
    ChartCalendar.GetCalendarWidthValue(ChartObj.DAY_OF_YEAR, 0.75), _
    attrib1, openHighAttrib, closeHighAttrib)
thePlot1.SetFastClipMode(ChartObj.FASTCLIP_X)
chartVu.AddChartObject(thePlot1)

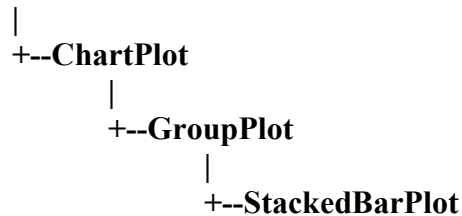
```

* Note how the **ChartCalendar.GetCalendarWidthValue** method calculates the width of the bars as a function of time, in this case a width of 0.75 days.

Stacked Bar Plots

Class StackedBarPlot

GraphObj



The **StackedBarPlot** class extends the **GroupPlot** class and displays data in stacked bar format. In a stacked bar plot each group is stacked on top of one another, each group bar a cumulative sum of the related group items before it. . When used with a 2½ D dataset, the position of the each bar with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property specifies the 3D depth of the bar. When a stacked bar plot is attached to a 3D dataset, each bar is not restricted to a z-axis plane and can traverse 3D space unrestricted. The individual elements of the bar plot are normally 3D bars, but you can also set the bar plots **BarShapeOption** property to a cylinder (CYLINDER3D).



*Stacked bar plots (Bargraphs.GroupBargraphs)***StackedBarPlot constructor**

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal rbarwidth As Double, _
    ByVal rbarbase As Double, _
    ByVal attribs As ChartAttribute(), _
    ByVal nbarjust As Integer _
)

[C#]
public StackedBarPlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    double rbarwidth,
    double rbarbase,
    ChartAttribute[] attribs,
    int nbarjust
);

```

<i>transform</i>	The coordinate system for the new StackedBarPlot object.
<i>dataset</i>	The stacked bar graph represents the values in this group dataset.
<i>rbarwidth</i>	The width of the stacked bars in units of the independent axis.
<i>rbarbase</i>	The stacked bars start at the value <i>rbarbase</i> , and extend to the group bar values represented by the dataset.
<i>attribs</i>	An array of ChartAttribute objects, sized the same as the number of groups in the dataset, that specify the attributes (outline color and fill color) for each group of a stacked bar graph.
<i>nbarjust</i>	The stacked bars are justified with respect to the x-values in the dataset using the <i>rbarjust</i> justification value (JUSTIFY_MIN, JUSTIFY_CENTER, or JUSTIFY_MAX).

Each stacked bar can be labeled with the group value bar using the bar data point methods, see the example below.

The attributes for each group can set or modified using the **SetSegment...** methods, where the segment number parameter cooresponds to the group number. These methods include **SetSegmentAttributes**, **SetSegmentFillColor**, **SetSegmentLineColor**, and **SetSegmentColor**.

Stacked bar plot example (extracted from the example program Bargraphs.GroupBargraphs)

[C#]

```
TimeCoordinates pTransform2 = new TimeCoordinates();
// User same dataset as Group bar plot, set stacked mode flag
Dataset1.SetStackMode(ChartObj.AUTOAXES_STACKED);
pTransform2.AutoScale(Dataset1,ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR);

pTransform2.SetTimeScaleStart(new ChartCalendar(1997,ChartObj.JANUARY,1));
pTransform2.SetTimeScaleStop(new ChartCalendar(2003,ChartObj.JANUARY,1));
pTransform2.SetGraphBorderDiagonal(0.55, .1, .95, 0.75) ;

pTransform2.SetScaleStartY(0);
.
. // Define axes, axes labels, and grids
.
StackedBarPlot thePlot2 =
    new StackedBarPlot(pTransform2, Dataset1,
        ChartCalendar.GetCalendarWidthValue(ChartObj.YEAR,0.75), 0.0,
        attribArray, ChartObj.JUSTIFY_CENTER);
NumericLabel bardatavalue = thePlot2.GetPlotLabelTemplate();
bardatavalue.SetTextFont(theFont);
bardatavalue.SetNumericFormat(ChartObj.CURRENCYFORMAT);
bardatavalue.SetDecimalPos(1);
bardatavalue.SetColor(Color.Black);
thePlot2.SetPlotLabelTemplate(bardatavalue);
thePlot2.SetBarDatapointLabelPosition(ChartObj.CENTERED_BAR);
thePlot2.SetShowDatapointValue(true);
chartVu.AddChartObject(thePlot2);
```

[Visual Basic]

```
` Stacked Bar Graph
    ' DEFINE SECOND GRAPH AS A STACKED BAR GRAPH
' Create coordinate system for second graph
Dim pTransform2 As New TimeCoordinates()
' User same dataset as Group bar plot, set stacked mode flag
Dataset1.SetStackMode(ChartObj.AUTOAXES_STACKED)
pTransform2.AutoScale(Dataset1, ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR)
```

```

' force x-axis time scale to a specific range
pTransform2.SetTimeScaleStart(New ChartCalendar(1997, ChartObj.JANUARY, 1))
pTransform2.SetTimeScaleStop(New ChartCalendar(2003, ChartObj.JANUARY, 1))
' force starting position of y-scale to 0
pTransform2.SetScaleStartY(0)

' define position of chart in graph area
pTransform2.SetGraphBorderDiagonal(0.1, 0.55, 0.9, 0.88)

' Set chart rotation
protate = New Point3D(10, 20, 0)
pTransform2.AbsRotateCoordinateSystem(protate)
chartVu.SetFractionalZViewportDepth(0.1)
.
.
.

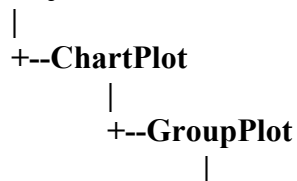
' create a stacked bar plot
Dim thePlot2 As New StackedBarPlot(pTransform2, Dataset1, _
    ChartCalendar.GetCalendarWidthValue(ChartObj.YEAR, 0.75), 0.0, _
    attribArray, ChartObj.JUSTIFY_CENTER)
Dim bardatavalue As NumericLabel = thePlot2.GetPlotLabelTemplate()
' Define internal data point labeling of the bars
bardatavalue.SetTextFont(theFont)
bardatavalue.SetNumericFormat(ChartObj.CURRENCYFORMAT)
bardatavalue.SetDecimalPos(1)
bardatavalue.SetColor(Color.Black)
thePlot2.SetPlotLabelTemplate(bardatavalue)
thePlot2.SetBarDatapointLabelPosition(ChartObj.CENTERED_BAR)
thePlot2.SetShowDatapointValue(True)
chartVu.AddChartObject(thePlot2)

```

Stacked Line Plots

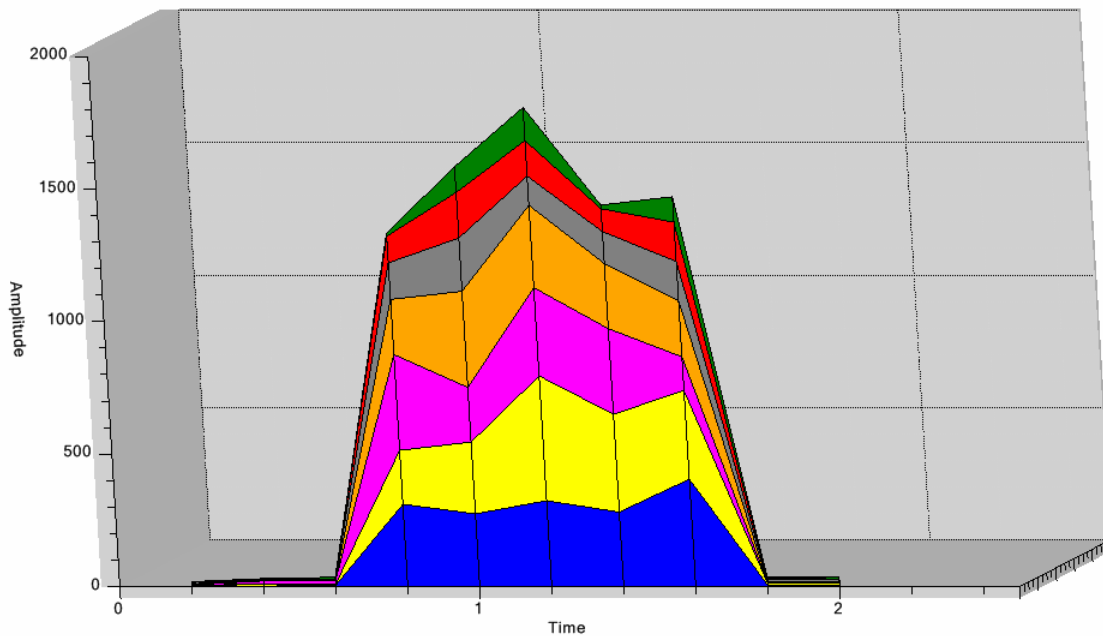
Class StackedLinePlot

GraphObj



+--StackedLinePlot

The **StackedLinePlot** class extends the **GroupPlot** class and displays data in stacked line format. In a stacked line plot each group is stacked on top of one another, each group line a cumulative sum of the groups before it. When used with a 2½ D dataset, the position of the each bar with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property is ignored since a line plot has no depth. When a stacked bar plot is attached to a 3D dataset, each bar is not restricted to a z-axis plane and can traverse 3D space unrestricted.



Stacked line plots (Same as MultiLinePlots.StackedLines)

StackedLinePlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal attrs As ChartAttribute() _
)

[C#]
public StackedLinePlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    ChartAttribute[] attrs
);
```

transform

The coordinate system for the new **StackedLinePlot** object.

<i>dataset</i>	The stacked line plot represents the values in this group dataset.
<i>attribs</i>	An array of ChartAttribute objects, sized the same as the number of groups in the dataset specify the attributes (line color and line style) for each group of the stacked line graph.

The attributes for each group can set or modified using the **SetSegment...** methods, where the segment number parameter corresponds to the group number. These methods include **SetSegmentAttributes**, **SetSegmentFillColor**, **SetSegmentLineColor**, and **SetSegmentColor**.

Stacked line plot example (extracted from the example program **MultiLinePlots.StackedLines**)

[C#]

```
int numPoints = 100;
int numGroups = 7;
double []x1 = new double[numPoints];
double [,]y1 = new double[numGroups,numPoints];

.
. // Initialize data
.

theFont = new Font("SansSerif", 10, FontStyle.Bold);

GroupDataset Dataset1 = new GroupDataset("First",x1,y1);
Dataset1.SetStackMode(ChartObj.AUTOAXES_STACKED);
CartesianCoordinates pTransform1 =
    new CartesianCoordinates( ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
pTransform1.SetScaleStartX(0);
pTransform1.SetScaleStartY(0);

Background background = new Background( pTransform1, ChartObj.PLOT_BACKGROUND,
    Color.FromArgb(255,255,255));
chartVu.AddChartObject(background);

pTransform1.SetGraphBorderDiagonal(0.15, .15, .90, 0.75) ;

.
. // Define axes, axes labels and grids
```

```

.

ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid);
attrib1.SetFillFlag(true);
attrib1.SetLineFlag(false);
ChartAttribute []attribArray = new ChartAttribute[numGroups];
for (i=0; i < numGroups; i++)
    attribArray[i] = (ChartAttribute) attrib1.Clone();
attribArray[0].SetFillColor(Color.Blue);
attribArray[1].SetFillColor(Color.Yellow);
attribArray[2].SetFillColor(Color.Magenta);
attribArray[3].SetFillColor(Color.Orange);
attribArray[4].SetFillColor(Color.Gray);
attribArray[5].SetFillColor(Color.Red);
attribArray[6].SetFillColor(Color.Green);
StackedLinePlot thePlot1 =
    new StackedLinePlot(pTransform1, Dataset1, attribArray);
chartVu.AddChartObject(thePlot1);

```

[Visual Basic]

```

' Simulate data
Dim numPoints As Integer = 10
Dim numGroups As Integer = 7
Dim x1(numPoints - 1) As Double
Dim y1(numGroups - 1, numPoints - 1) As Double

.
. // Initialize data
.

' Define a group dataset for the stacked line plot
Dim Dataset1 As New GroupDataset("First", x1, y1)
Dataset1.ImplicitDepthValue = 0.2
Dataset1.ImplicitZValue = 0.5
Dataset1.SetStackMode(ChartObj.AUTOAXES_STACKED)
' Define coordinate system for stacked line plot
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE,
ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
' Force starting values of coordinate system to 0
pTransform1.SetScaleStartX(0)
pTransform1.SetScaleStartY(0)

```

```

' Set rotate parameters
pTransform1.AbsRotateCoordinateSystem(New Point3D(10, 20, 0))

' Set chart depth
chartVu.SetFractionalZViewportDepth(0.3)

.
.
.
.

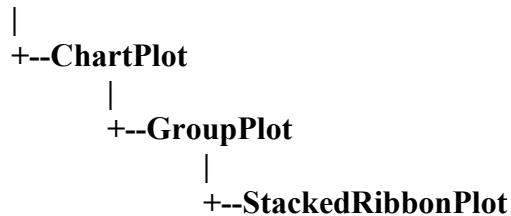
' Define attributes for stacked line plot
Dim attrib1 As New ChartAttribute(Color.Black, 1, DashStyle.Solid, Color.Blue)
Dim attribArray(numGroups) As ChartAttribute
For i = 0 To numGroups - 1
    attribArray(i) = CType(attrib1.Clone(), ChartAttribute)
Next i
attribArray(0).SetFillColor(Color.Blue)
attribArray(1).SetFillColor(Color.Yellow)
attribArray(2).SetFillColor(Color.Magenta)
attribArray(3).SetFillColor(Color.Orange)
attribArray(4).SetFillColor(Color.Gray)
attribArray(5).SetFillColor(Color.Red)
attribArray(6).SetFillColor(Color.Green)

' Define a stacked line plot
Dim thePlot1 As New StackedLinePlot(pTransform1, Dataset1, attribArray)
chartVu.AddChartObject(thePlot1)

```

Class StackedRibbonPlot

GraphObj



The **StackedRibbonPlot** class extends the **GroupPlot** class and displays data in stacked line format. In a stacked ribbon plot each group is stacked on top of one another, each group ribbon a cumulative sum of the groups before it. The stacked ribbon plot is used with a 2½ D dataset, the position of the each ribbon with respect to the z-axis is set using

the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** sets the 3D depth of the ribbon.

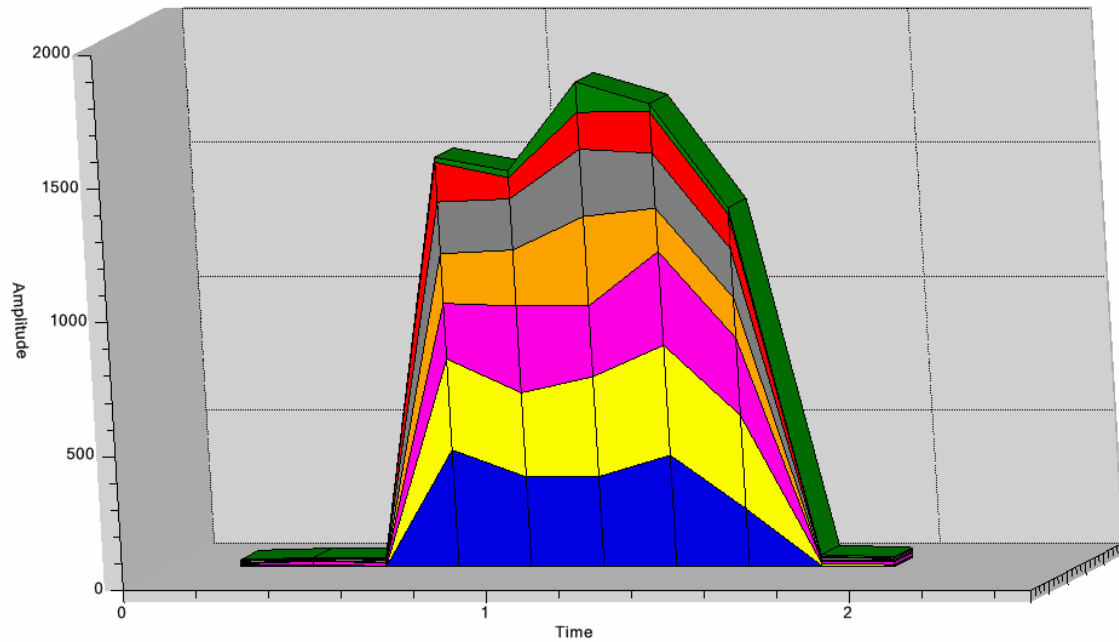
StackedLinePlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As GroupDataset, _
    ByVal attrs As ChartAttribute() _
)

[C#]
public StackedLinePlot(
    PhysicalCoordinates transform,
    GroupDataset dataset,
    ChartAttribute[] attrs
);
```

<i>transform</i>	The coordinate system for the new StackedLinePlot object.
<i>dataset</i>	The stacked line plot represents the values in this group dataset.
<i>attrs</i>	An array of ChartAttribute objects, sized the same as the number of groups in the dataset specify the attributes (line color and line style) for each group of the stacked line graph.

The attributes for each group can set or modified using the **SetSegment...** methods, where the segment number parameter corresponds to the group number. These methods include **SetSegmentAttributes**, **SetSegmentFillColor**, **SetSegmentLineColor**, and **SetSegmentColor**.



Stacked ribbon plots (MultiLinePlots.StackedLines)

**Stacked line plot example (extracted from the example program
MultiLinePlots.StackedLines)**

[C#]

```

.
.
.
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid);
attrib1.SetFillFlag(true);
attrib1.SetLineFlag(false);
ChartAttribute []attribArray = new ChartAttribute[numGroups];
for (i=0; i < numGroups; i++)
    attribArray[i] = (ChartAttribute) attrib1.Clone();
attribArray[0].SetFillColor(Color.Blue);
attribArray[1].SetFillColor(Color.Yellow);
attribArray[2].SetFillColor(Color.Magenta);
attribArray[3].SetFillColor(Color.Orange);
attribArray[4].SetFillColor(Color.Gray);
attribArray[5].SetFillColor(Color.Red);
attribArray[6].SetFillColor(Color.Green);
StackedRibbonPlot thePlot1 =

```

```

        new StackedRibbonPlot(pTransform1, Dataset1, attribArray);
chartVu.AddChartObject(thePlot1);

```

[Visual Basic]

```

' Define attributes for stacked line plot
Dim attrib1 As New ChartAttribute(Color.Black, 1, DashStyle.Solid, Color.Blue)
Dim attribArray(numGroups) As ChartAttribute
For i = 0 To numGroups - 1
    attribArray(i) = CType(attrib1.Clone(), ChartAttribute)
Next i
attribArray(0).SetFillColor(Color.Blue)
attribArray(1).SetFillColor(Color.Yellow)
attribArray(2).SetFillColor(Color.Magenta)
attribArray(3).SetFillColor(Color.Orange)
attribArray(4).SetFillColor(Color.Gray)
attribArray(5).SetFillColor(Color.Red)
attribArray(6).SetFillColor(Color.Green)

' Define a stacked line plot
StackedRibbonPlot thePlot1 = new StackedRibbonPlot(pTransform1, _
    Dataset1, attribArray);

```


12. Contour Plotting

ChartPlot

ContourPlot

The **ContourPlot** class displays contour data organized in a **ContourDataset**.dataset. The line contour graph draws continuous lines through the data at xy-values representing equal values of z, analogous to the equal pressure lines (isobars) of a weather map. A filled contour graph fills the area between two contour levels with a specific color.

Line and Filled Contour Plots

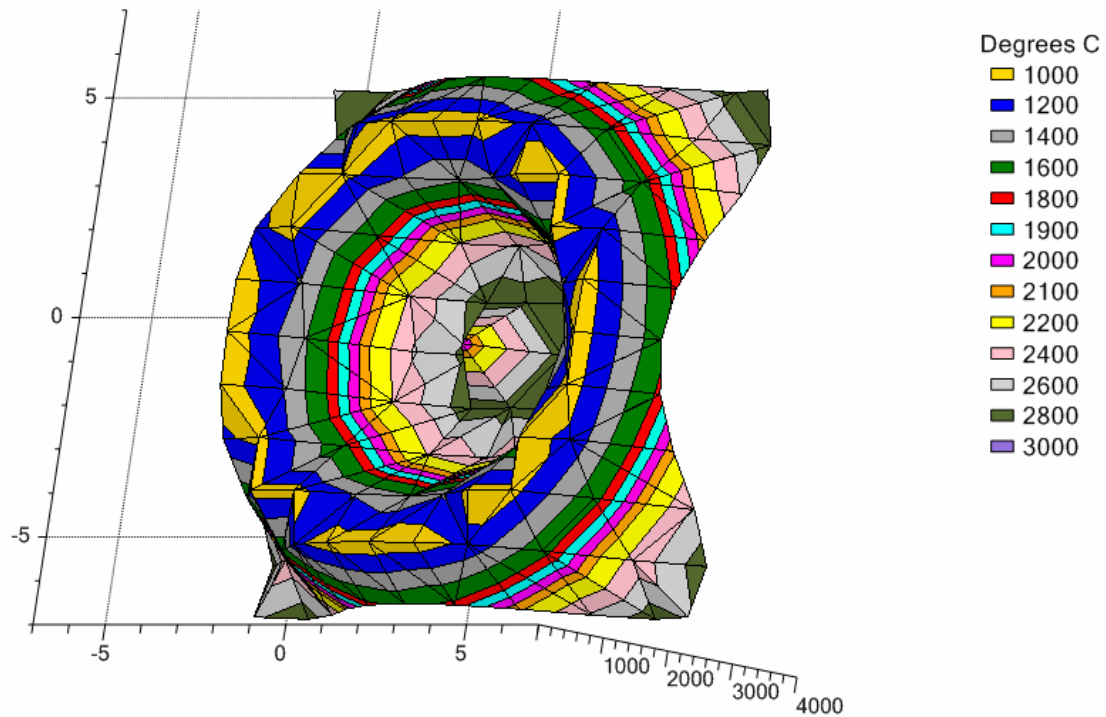
Class ContourPlot

GraphObj

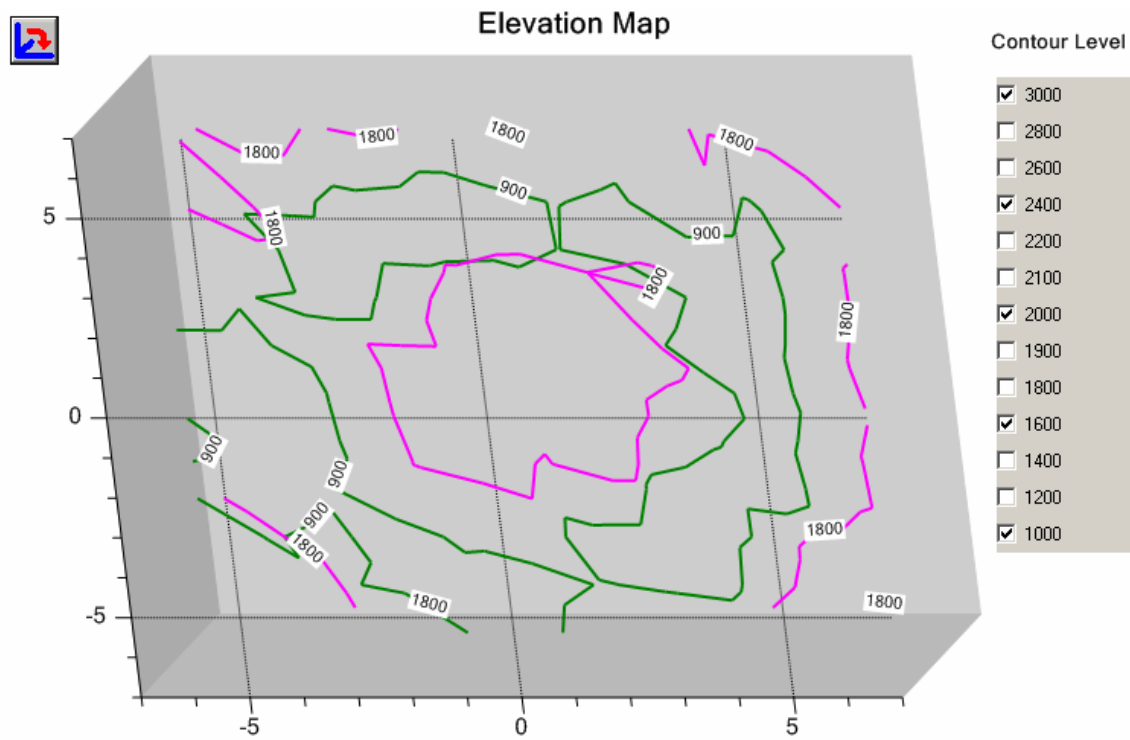
|
+--**ChartPlot**

|
+--**ContourPlot**

The **ContourPlot** class is a subclass of the **ChartPlot** class and displays a contour plot using either lines, or regions filled with color. The two constructors below differ only by the inclusion of the contour line flags and the contour label flags in the second constructor.



Contour plot (*ContourPlots.ContourFillPlot*)



The contour lines can be labeled when not too close together.

*Contour line plot (ContourPlots.ContourLinePlot)***ContourPlot constructors**

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As ContourDataset, _
    ByVal contourlevels As Double(), _
    ByVal attribs As ChartAttribute(), _
    ByVal numcontourlevels As Integer, _
    ByVal contourtype As Integer _
)

Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As ContourDataset, _
    ByVal contourlevels As Double(), _
    ByVal attribs As ChartAttribute(), _
    ByVal blineflags As Boolean(), _
    ByVal blabelflags As Boolean(), _
    ByVal numcontourlevels As Integer, _
    ByVal contourtype As Integer _
)

[C#]
public ContourPlot(
    PhysicalCoordinates transform,
    ContourDataset dataset,
    double[] contourlevels,
    ChartAttribute[] attribs,
    int numcontourlevels,
    int contourtype
);

public ContourPlot(
    PhysicalCoordinates transform,
    ContourDataset dataset,
    double[] contourlevels,
    ChartAttribute[] attribs,
    bool[] blineflags,
    bool[] blabelflags,
    int numcontourlevels,
    int contourtype
);

```

<i>transform</i>	The coordinate system for the new ContourPlot object.
<i>dataset</i>	The ContourDataset plot will represent the xyz values in this contour data set.
<i>contourlevels</i>	An array, size [numcontourlevels], of the contour levels used in the contour plot.
<i>attribs</i>	An array of color and fill attributes, size [numcontourlevels+1]. If the <i>contourtype</i> is CONTOUR_LINE, the colors of elements 0..numcontourlevels-1 set the colors of the contour lines. If the

	contourtype is CONTOUR_FILL, elements 0.. <i>numcontourlevels</i> set the colors of the contour regions.
<i>blinelflags</i>	An array, size [<i>numcontourlevels</i>], of boolean flags specifying whether a contour line should be displayed.
<i>blabelflags</i>	An array, size [<i>numcontourlevels</i>], of boolean flags specifying whether a contour line should be labeled with the numeric value of the associated contour level.
<i>numcontourlevels</i>	The number of contour levels.
<i>contourtype</i>	Specifies if the contour plot uses contour lines (CONTOUR_LINE), filled contour regions (CONTOUR_FILL) or both (CONTOUR_LINEANDFILL).

Contour plots that use the CONTOUR_FILL algorithm require one more color than the CONTOUR_LINE algorithm.

The attributes for each contour line can set or modified using the **SetSegment...** methods, where the segment number parameter cooresponds to the contour value index. These methods include **SetSegmentAttributes**, **SetSegmentFillColor**, **SetSegmentLineColor**, and **SetSegmentColor**.

Contour line plot example (extracted from the example program ContourPlots.ContourLinePlot)

[C#]

```
double []contourlevels = {0,300, 600, 900, 1200, 1500, 1800, 2100, 2400, 2700,
3000, 3300, 3600};

.
.
.

pTransform1 = new CartesianCoordinates(-7, -7, 0, 7, 7, 3000);

pTransform1.SetGraphBorderDiagonal(0.10, .10, .8, 0.85) ;

CreateRegularGridPolysurface();

.
.
```



```

.

ChartAttribute []attribs = new ChartAttribute[numcontourlevels+1];
for (i=0; i <= numcontourlevels; i++)
{
    Color color = Color.FromArgb((int) (255* ChartSupport.GetRandomDouble()),
        (int) (255 * ChartSupport.GetRandomDouble()),
        (int) (255 * ChartSupport.GetRandomDouble()));
    attribs[i] = new ChartAttribute(Color.Black,1,DashStyle.Solid,color);
    attribs[i].SetFillFlag(true);
    attribs[i].SetLineFlag(true);
    attribs[i].SymbolSize = 16;
}
attribs[0].SetColor(Color.Black);
attribs[1].SetColor(Color.Blue);
attribs[2].SetColor(Color.DarkGray);
attribs[3].SetColor(Color.Green);
attribs[4].SetColor(Color.Red);
attribs[5].SetColor(Color.Cyan);
attribs[6].SetColor(Color.Magenta);
attribs[7].SetColor(Color.Orange);
attribs[8].SetColor(Color.Yellow);
attribs[9].SetColor(Color.Chartreuse);
attribs[10].SetColor(Color.MediumVioletRed);
attribs[11].SetColor(Color.Salmon);
attribs[12].SetColor(Color.DarkKhaki);

.
.
.

thePlot1 = new ContourPlot(pTransform1, dataset1, contourlevels, attribs,
    lineflags, labelflags, numcontourlevels, ChartObj.CONTOUR_LINE);

thePlot1.SetPolygonGridOn(true);
thePlot1.SetContourLineAlgorithm(ChartObj.CONTOUR_LINEWALK);
NumericLabel contourlabel = thePlot1.GetPlotLabelTemplate();
Font contourlabelFont = new Font("Microsoft Sans Serif", 8, FontStyle.Regular);
contourlabel.SetDecimalPos(0);
contourlabel.SetTextFont(contourlabelFont);
contourlabel.TextBgMode = true;
contourlabel.TextBgColor = Color.White;
thePlot1.SetPlotLabelTemplate(contourlabel);

```

```
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
Dim contourlevels As Double() = {0, 300, 600, 900, 1200, 1500, 1800, 2100, 2400,  
2700, 3000, 3300, 3600}
```

```
.  
.  
.
```

```
pTransform1 = New CartesianCoordinates(-7, -7, 0, 7, 7, 3000)
```

```
pTransform1.SetGraphBorderDiagonal(0.1, 0.1, 0.8, 0.85)
```

```
CreateRegularGridPolysurface()
```

```
.  
.  
.
```

```
Dim attribs(numcontourlevels) As ChartAttribute
```

```
For i = 0 To numcontourlevels
```

```
    Dim color As Color = Color.FromArgb(CInt(255 *  
ChartSupport.GetRandomDouble()), CInt(255 * ChartSupport.GetRandomDouble()),  
CInt(255 * ChartSupport.GetRandomDouble()))
```

```
    attribs(i) = New ChartAttribute(color, 2, DashStyle.Solid, color)
```

```
    attribs(i).SetFillFlag(True)
```

```
Next i
```

```
attribs(0).SetColor(Color.Black)
```

```
attribs(1).SetColor(Color.Blue)
```

```
attribs(2).SetColor(Color.DarkGray)
```

```
attribs(3).SetColor(Color.Green)
```

```
attribs(4).SetColor(Color.Red)
```

```
attribs(5).SetColor(Color.Cyan)
```

```
attribs(6).SetColor(Color.Magenta)
```

```
attribs(7).SetColor(Color.Orange)
```

```
attribs(8).SetColor(Color.Yellow)
```

```
attribs(9).SetColor(Color.Chartreuse)
```

```
attribs(10).SetColor(Color.MediumVioletRed)
```

```

attribs(11).SetColor(Color.Salmon)
attribs(12).SetColor(Color.DarkKhaki)
.
.
.
thePlot1 = New ContourPlot(pTransform1, dataset1, contourlevels, _
    attribs, lineflags, labelflags, numcontourlevels, ChartObj.CONTOUR_LINE)
thePlot1.SetPolygonGridOn(True)
thePlot1.SetContourLineAlgorithm(ChartObj.CONTOUR_LINEWALK)
Dim contourlabel As NumericLabel = thePlot1.GetPlotLabelTemplate()
Dim contourLabelFont As New Font("Microsoft Sans Serif", 8, FontStyle.Regular)
contourlabel.SetDecimalPos(0)
contourlabel.SetTextFont(contourLabelFont)
contourlabel.TextBgMode = True
contourlabel.TextBgColor = Color.White
thePlot1.SetPlotLabelTemplate(contourlabel)
chartVu.AddChartObject(thePlot1)

```


13. Data Markers and Data Cursors

Marker

DataCursor

Data markers are symbols and lines that can be “dropped” on to the data presented in a graph, much like a bookmark in a word processing document. Place the markers in a chart under program control or in response to a mouse event in the graph window. Data markers can be positioned in 3D space.

Data cursors are temporary lines or symbols, drawn using the XOR drawing mode, that are used to help position the mouse cursor over the desired section of a graph. Standard data cursors include cross hairs, a box, and horizontal and/or vertical lines. Data cursors are restricted to the XY plane. The z-value of the data cursor is specified using the data cursors **ImplicitZValue** property.

Data Markers

Class Marker

GraphObj

|
+--**Marker**

Create data markers using the **Marker** class. The constructor below creates a new **Marker** object using the specified coordinate system, marker type, marker position and marker size. The marker is drawn facing the XY plane at the specified x, y and z position.

Marker constructor

[Visual Basic]

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal nmarkertype As Integer, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal z As Double, _
    ByVal rsize As Double, _
    ByVal npostype As Integer _
)

Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal nmarkertype As Integer, _
    ByVal x As Double, _
    ByVal y As Double, _
```

```

        ByVal rsize As Double, _
        ByVal npostype As Integer _
    )

    [C#]
    public Marker(
        PhysicalCoordinates transform,
        int nmarkertype,
        double x,
        double y,
        double z,
        double rsize,
        int npostype
    );

    [C#]public Marker(
        PhysicalCoordinates transform,
        int nmarkertype,
        double x,
        double y,
        double rsize,
        int npostype
    );

```

<i>transform</i>	Places the marker in the coordinate system defined by transform.
<i>nmarkertype</i>	Specifies the shape of the current chart marker. Use one of the chart marker constants: <code>MARKER_NULL</code> , <code>MARKER_VLINE</code> , <code>MARKER_HLINE</code> , <code>MARKER_CROSS</code> , <code>MARKER_BOX</code> or <code>MARKER_HVLINE</code> .
<i>x</i>	Specifies the x-value of the marker position
<i>y</i>	Specifies the y-value of the marker position
<i>z</i>	Specifies the z-value of the marker position
<i>rsize</i>	Specifies the size of the cross hair marker (<code>MARKER_CROSS</code>) and the box marker (<code>MARKER_BOX</code>) in .Net device coordinates.
<i>npostype</i>	Specifies the if the position of the marker is specified in physical coordinates, normalized coordinates or .Net device coordinates. Use one of the position constants: <code>DEV_POS</code> , <code>PHYS_POS</code> , <code>NORM_GRAPH_POS</code> , <code>NORM_PLOT_POS</code> .

The marker constants signify:

<code>MARKER_NULL</code>	An invisible marker

MARKER_VLINE	The marker is a vertical line extending from the top of the plot area volume to the bottom, passing through the x-value of the marker position.
MARKER_HLINE	The marker is a horizontal line extending from the left of the plot area volume to the right, passing through the y-value of the marker position.
MARKER_HVLINE	The marker combines both MARKER_VLINE and MARKER_HLINE, marking the data point with horizontal and vertical lines.
MARKER_CROSS	The marker is a cross hair centered on the marker position. Set the size of the cross hair using .Net device coordinates, in the object constructor, or later using the setMarkerSize method.
MARKER_BOX	The marker is a box centered on the marker position. Set the size of the box using .Net device coordinates, in the object constructor, or later using the setMarkerSize method.

Drop a marker anywhere on a plot by specifying the coordinates. The example below places a 10 pixel wide marker in the center of the plot area volume.

Simple marker example

[C#]

```

CartesianCoordinates pTransform1 =
    new CartesianCoordinates( 0.0, 0.0,0.0, 10.0, 20.0, 1.0);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
.
.
.
double xpos = 5.0;
double ypos = 10.0;
double zpos = 0.0;
Marker amarker = new Marker(pTransform1, ChartObj.MARKER_BOX, xpos, ypos, zpos,
    10, ChartObj.PHYS_POS);
theChartView.AddChartObject(amarker);

```

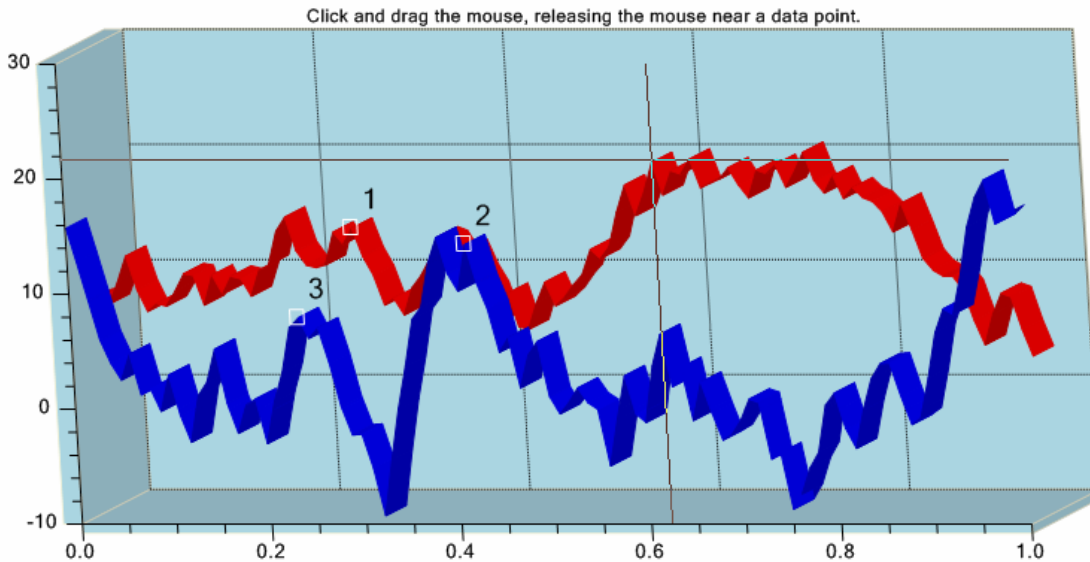
[Visual Basic]

```

Dim pTransform1 As CartesianCoordinates = _
    New CartesianCoordinates( 0.0, 0.0, 0.0, 10.0, 20.0, 1.0)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
.
.
.
Dim xpos As Double = 5.0
Dim ypos As Double = 10.0
Dim zpos As Double = 0.0
Dim amarker As Marker = New Marker(pTransform1, ChartObj.MARKER_BOX, _
    xpos, ypos, zpos, 10, ChartObj.PHYS_POS)
theChartView.AddChartObject(amarker)

```

See the example **MouseListeners.DataCursors** for a more complex example combining data cursors and markers.

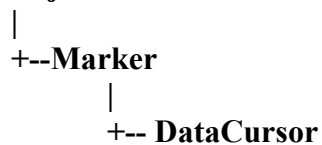


MouseListeners.DataCursors

Data Cursors

Class DataCursor

GraphObj



Data cursors are an extension of the marker class. Data cursors combine the .Net mouse event delegates with the **Marker** class, creating a marker that tracks the mouse and updates dynamically using an XOR drawing mode. This constructor creates a new **DataCursor** object using the specified coordinate system, marker type and marker size. Data cursors are restricted to the XY plane. The z-value of the data cursor is specified using the data cursors **ImplicitZValue** property.

DataCursor constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal component As ChartView, _
    ByVal transform As PhysicalCoordinates, _
    ByVal zpos As Double, _
    ByVal nmarkertype As Integer, _
    ByVal rsize As Double _
)

[C#]
public DataCursor(
    ChartView component,
    PhysicalCoordinates transform,
    double zpos,
    int nmarkertype,
    double rsize
);
```

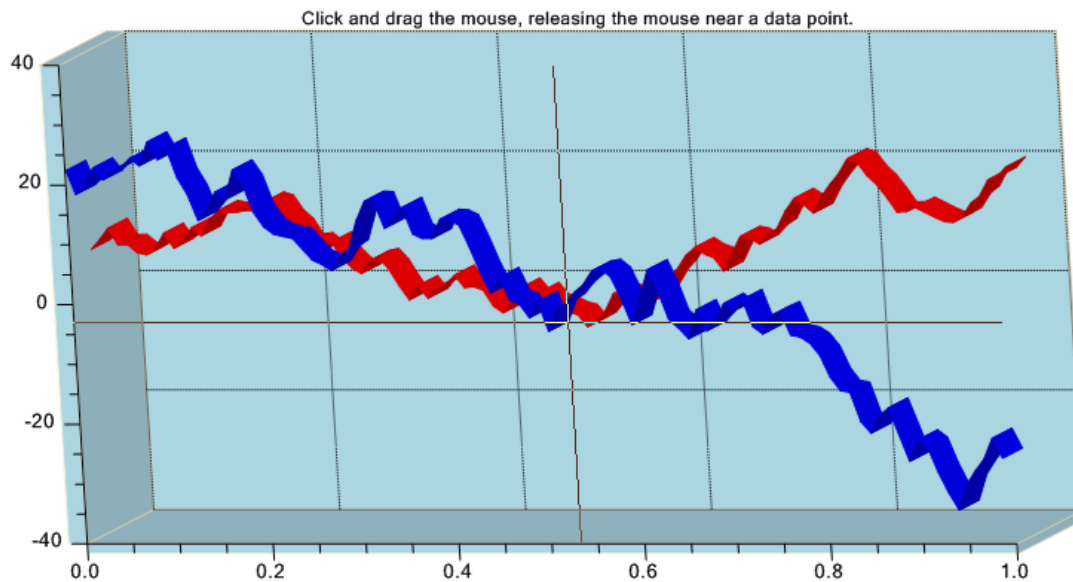
<i>component</i>	A reference to the ChartView object that the chart is placed in.
<i>transform</i>	The PhysicalCoordinates object associated with the data cursor.
<i>zpos</i>	The initial z-position of the data cursor in the xy plane.
<i>nmarkdertype</i>	The marker type. Use one of the Marker marker type constants: MARKER_VLINE .. MARKER_BOX .
<i>rsize</i>	The size in .Net device coordinates of the MARKER_BOX and MARKER_CROSS style cursors.

See the **Marker** constructor description for more information about the **Marker** type constants.

Create the **DataCursor** object and then install it using the **ChartView.SetCurrentMouseListener** method. This adds the **DataCursor** object as a **MouseListener** to the **ChartView** object. Enable/Disable the function using **DataCursor.SetEnable** method. Call **DataCursor.SetCurrentMouseListener(null)** to remove the object as a mouse listener for the chart view.

Since the **DataCursor** class implements mouse event delegates, it has methods implementing the mouse events **OnMouseMove**, **OnDoubleClick**, **OnClick**, **OnMouseDown**, and **OnMouseUp**. The default usage of the **DataCursor** class creates the data marker when the mouse is pressed. As long as the mouse is pressed, the data cursor tracks the mouse position. Release the mouse button and the marker disappears. When using data markers it is often desirable to do additional things during the mouse events. In this case, you derive a new class from **DataCursor**, override the mouse events that you want to intercept, and add your own code to these events. Make sure you call the parents version of the same mouse event function so that the data cursor continues to track the mouse.

Simple data cursor example (Extracted from the **DataCursorsAndMarkers.DataCursors** example program)



DataCursorsAndMarkers.DataCursors

[C#]

```
// Create a standard data cursor object and add it as a mouse listener
DtaCursor dataCursorObj = new DataCursor( chartVu, pTransform1,
ChartObj.MARKER_HVLINE, 16.0);
dataCursorObj.ChartObjClipping = ChartObj.GRAPH_AREA_CLIPPING;
dataCursorObj.SetEnable(true);
chartVu.SetCurrentMouseListener(dataCursorObj);
```

[Visual Basic]

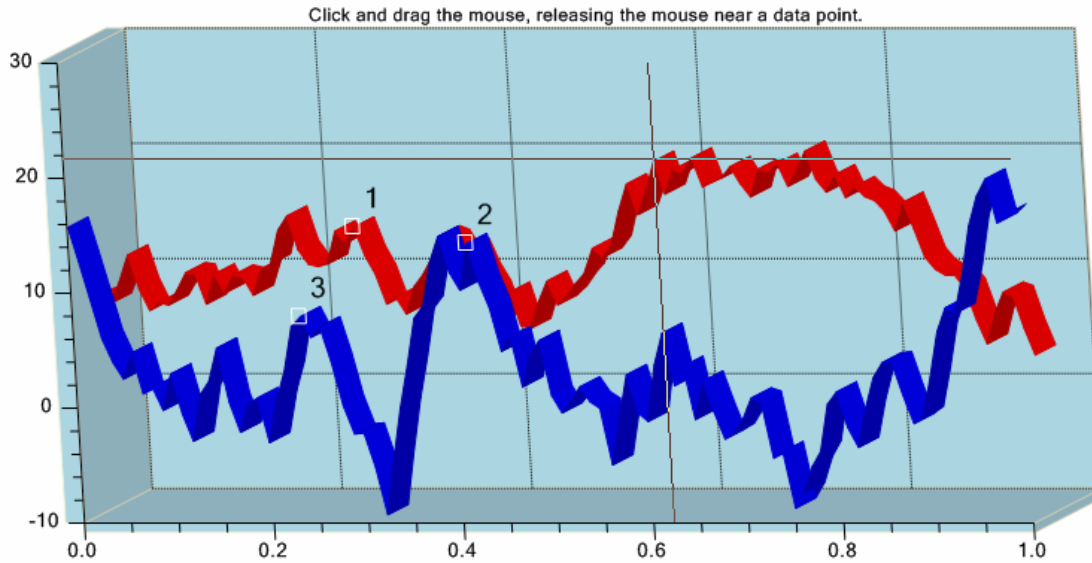
```
' Create a standard data cursor object and add it as a mouse listener
Dim dataCursorObj As New DataCursor(chartVu, pTransform1, _
    ChartObj.MARKER_HVLINE, 16.0)
dataCursorObj.ChartObjClipping = ChartObj.GRAPH_AREA_CLIPPING
dataCursorObj.SetEnable(True)
chartVu.SetCurrentMouseListener(dataCursorObj)
```

A marker can be placed at any xy coordinate location in a graph. It is often desirable to place a marker at the exact location of a data point in one of the datasets plotted in the graph. Many applications require the user to click on the approximate location of a point, and then the software must find the data point nearest that click and mark it. The **DataCursor** and **Marker** classes, in combination with the plot objects **CalcNearestPoint** methods, accomplish this. The **DataCursor** class positions the mouse cursor and retrieves the initial xy coordinates. The **CalcNearestPoint** method for each plot object (**SimpleLinePlot**, **ScatterPlot**, etc.) in the graph determines the nearest data point to the mouse cursor for that object. Once all the plot objects are checked the data point nearest the mouse cursor position is marked by placing a **Marker** object at that exact xy location.

The example **DataCursorsAndMarkers.Markers** extends the previous **Marker** and **DataCursor** examples. In that example, the **OnMouseUp** event of the subclassed **DataCursor** object processes the plot objects looking for the nearest point, and then places a **Marker** object and a numeric label at that point.

Marking a data point (Adapted from the DataCursorsAndMarkers.Markers example program)

```
{ See the actual program for the source. }
```



DataCursorsAndMarkers.DataCursors

Another common reason for locating a data point is to display information associated with that data point. A good example is stock market data. A typical stock market display is a one-month chart of daily closing values for one or more stocks. You want to be able to click on a point in the chart and have the open, high, low and closing value for that day displayed in a pop-up box. The example program `FinancialExamples.OHLCChart` demonstrates how to use a **CustomDataToolTip** object as a popup box to display this type of data. In another related example, the program `PieCharts.SimplePieChart` shows how to trap a click on a specific pie slice and display additional data for that slice using a **CustomDataToolTip** object.

14. Moving Chart Objects and Data Points

MoveObj MoveData

Many of the subclasses of **GraphObj** are moveable using the mouse. This includes the axis, legend, text, image and shape classes. If you add the necessary support to your program, you can click and drag the object around in the chart. This may or not be desirable, since a user can ruin a carefully constructed chart by dragging objects around. It is just an option though, that you can add to the program. When an object is moved, its movement is restricted to the XY plane. The new position of the object retains its starting z-position value.

It is also possible to select a single data point in a simple plot object (**SimpleLinePlot**, **SimpleBarPlot**, **SimpleLineMarkerPlot** and **SimpleScatterPlot**) and move it with a click and drag operation of the mouse. Again, it is an option that you can add to the program if you want.

Moving Chart Objects

Class MoveObj MouseListener | +--MoveObj

The **MoveObj** mouse listener traps a mouse pressed event and then searches through all of the **GraphObj** derived objects in the view. A rectangle highlights the first object that meets the filter criteria and intersects the mouse cursor. Hold the mouse button down and the rectangle tracks the mouse. Release the mouse button and the position of the graph object updates to reflect the new physical coordinates of the bounding rectangle. When an object is moved, its movement is restricted to the XY plane. The position of the object retains its starting z-position value.

If no *objectfilter* parameter is specified, the default object filter is “GraphObj”.

MoveObject constructors

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal component As ChartView, _
    ByVal buttonmask As MouseButton, _
    ByVal objectfilter As String _
)
```

```
Overloads Public Sub New( _
    ByVal component As ChartView, _
    ByVal buttonmask As MouseButtons _
)
```

```
[C#]
public MoveObj(
    ChartView component,
    MouseButtons buttonmask,
    string objectfilter
);
```

```
public MoveObj(
    ChartView component,
    MouseButtons buttonmask
);
```

<i>component</i>	A reference to the ChartView object that the chart is placed in.
<i>buttonmask</i>	Specifies the mouse button that is trapped to invoke a move.
<i>objectfilter</i>	The fully qualified class name of the base class that is used to filter the desired class objects. The string "ChartText" causes the routine to move only objects derived from the ChartText class. If you want to move only specific objects of a given class, create a special subclass of that class. Then create your moveable objects using that subclass. Then specify your class name, i.e. MyTextClass , using the string "MyTextClass".

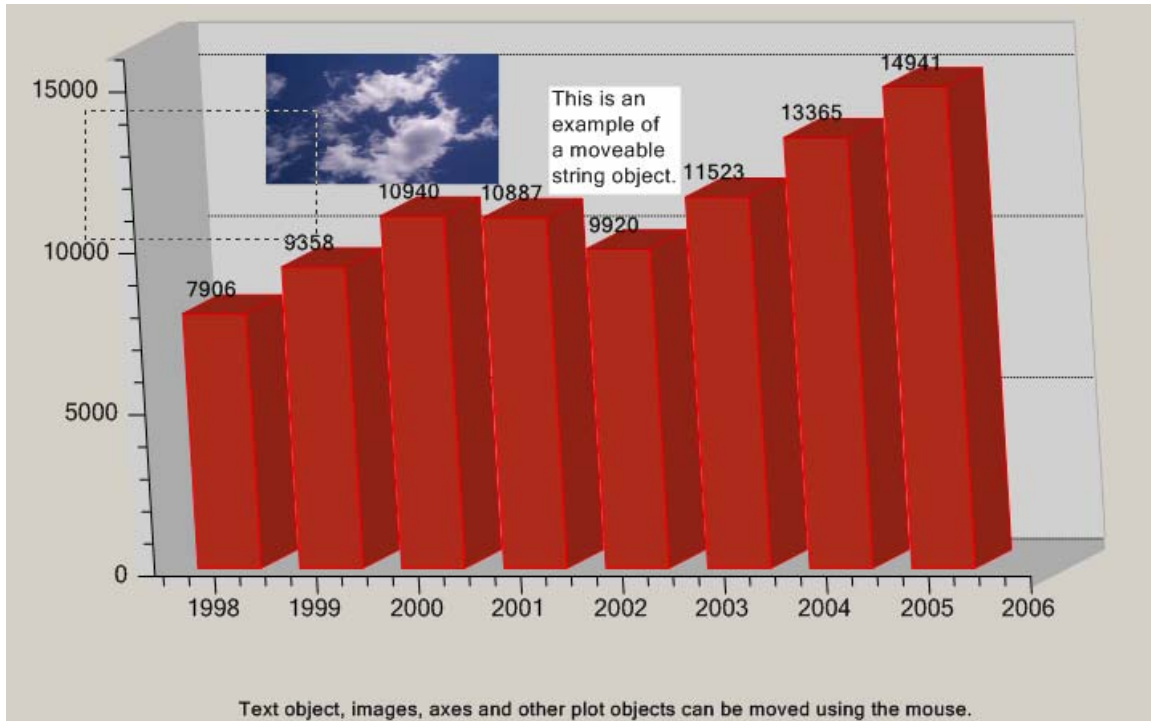
Create the **MoveObj** object and then install it using the **ChartView.SetCurrentMouseListener** method. This adds the **MoveObj** object as a **MouseListener** to the **ChartView** object. Enable/Disable the function using **MoveObj.SetEnable** method. Call **MoveObj.SetCurrentMouseListener(null)** to remove the object as a mouse listener for the chart view.

Not all **GraphObj** derived object are moveable. Call the **GraphObj.GetMoveableType** method and check to see if it returns **ChartObj.OBJECT_MOVEABLE**. Alternatively, you can call the **MoveObj.IsMoveableObject** method, passing in a reference to the object.

Most moveable objects move unrestricted in the x- and y direction. There are exceptions though. Axis objects move in the direction parallel to their current position, effectively changing the axis intercept, but not the extents of the axis endpoints. Axis labels always track their reference axis. The base axis defines the position of an **AxisTitle** text object and the chart view defines the position of a **ChartTitle** text object. Attempt to move these

objects and they revert to their original centered positions. If you require moveable chart and axis titles, use the generic **ChartText** class instead of the title classes.

Moving objects example (Extracted from the `MouseListeners.MoveObjects` class)



MouseListeners.MoveObjects

[C#]

```

ChartView chartVu = this;
.
.
.
// Define a dataset
TimeSimpleDataset Dataset1 = new TimeSimpleDataset("DJA",x1,y1, 0.2, 0.5);
// Create a compatible coordinate system for data
TimeCoordinates pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR , ChartObj.AUTOAXES_FAR);
.

```

```

.
.
// Add a simple mouse move listener
MoveObj mousetListener = new MoveObj(chartVu );
mousetListener.SetEnable(true);
mousetListener.SetMoveObjectFilter("GraphObj");
chartVu.SetCurrentMouseListener(mousetListener);

```

[Visual Basic]

```

Dim chartVu As ChartView = Me
.
.
.
' Define a dataset
Dim Dataset1 As New TimeSimpleDataset("DJA", x1, y1, 0.2, 0.5)
' Create a compatible coordinate system for data
Dim pTransform1 As New TimeCoordinates()
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
.
.
.
' Add a simple mouse move listener
Dim mousetListener As New MoveObj(chartVu)
mousetListener.SetEnable(True)
mousetListener.SetMoveObjectFilter("GraphObj")
chartVu.SetCurrentMouseListener(mousetListener)

```

Moving Simple Plot Object Data Points

Class MoveData

MouseListener

```

|
+---MoveData

```

The **MoveData** mouse listener traps a mouse pressed event and searches through all of the data points of the plot objects in the view. The data point closest to the mouse cursor location is compared against a threshold value, 10 device units, or pixels, by default. If the data point is within the threshold, and as long as the mouse button is held down, it tracks the mouse. Release the mouse button and the data value associated with the

selected data point updates to reflect the new physical coordinates of the data point, and the plot is redrawn. Since the algorithm searches through every data point of every plot object in the view, do not expect it to work particularly fast with millions or even thousands of data points. The practical number of data points that can be searched is obviously dependent on the speed of the host computer. When an object is moved, its movement is restricted to the XY plane. The position of the object retains its starting z-position value.

MoveData constructors

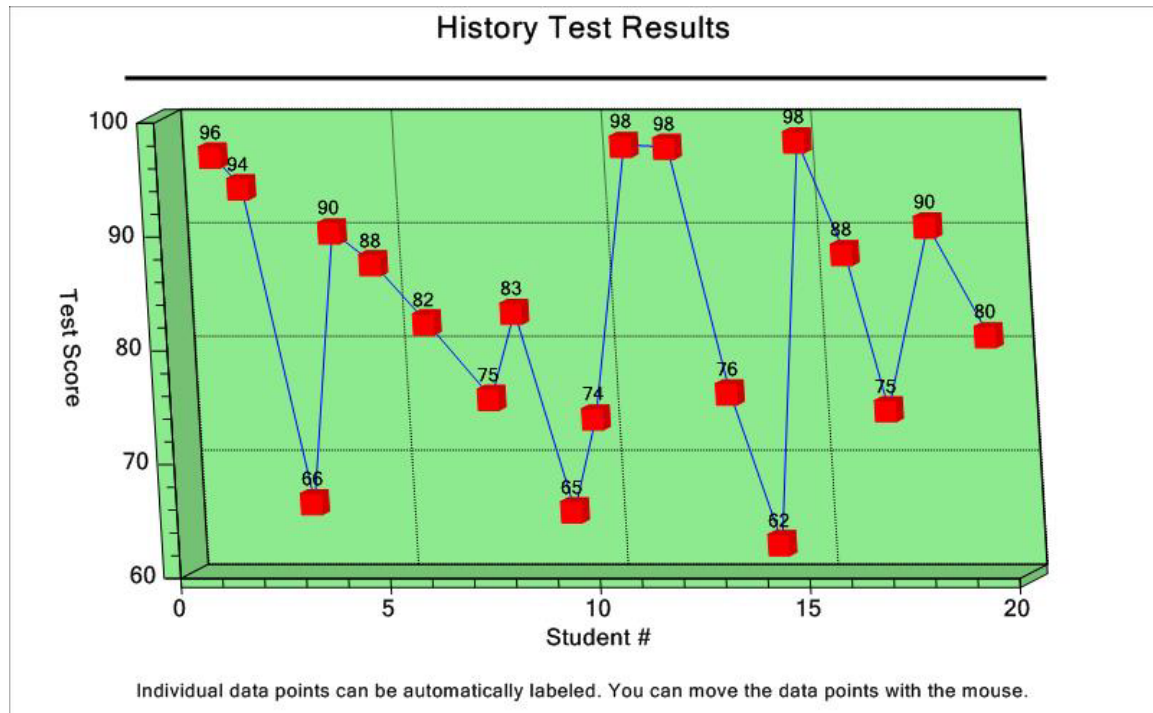
```
[Visual Basic]
Overloads Public Sub New( _
    ByVal component As ChartView, _
    ByVal transform As PhysicalCoordinates, _
    ByVal buttonmask As MouseButtons _
)
```

```
[C#]
public MoveData(
    ChartView component,
    PhysicalCoordinates transform,
    MouseButtons buttonmask
);
```

<i>component</i>	A reference to the ChartView object that the chart is placed in.
<i>transform</i>	The PhysicalCoordinates object associated with the MoveData object.
<i>buttonmask</i>	Specifies the mouse button that is trapped to invoke a move.

Create the **MoveData** object and then install it using the **ChartView.SetCurrentMouseListener** method. This adds the **MoveData** object as a **MouseListener** to the **ChartView** object. Enable/Disable the function using **MoveData.SetEnable** method. Call **MoveData.SetCurrentMouseListener(null)** to remove the object as a mouse listener for the chart view. Set the threshold distance for deciding if the nearest data point found is a “hit” using the **MoveData.SetHitTestThreshold** method.

Moving datapoints example (Extracted the example LabeledDatapoints.LabeledDatapoints). A more complicated example involving a custom mouse move object is found in the **MouseListeners.MoveDatapoints** example program.



LabeledDatapoints.LabeledDatapoints

[C#]

```

ChartView chartVu = this;
// Create a 3D dataset
ContourDataset Dataset1 = new ContourDataset("First",x1,y1, z1);
// Create a coordinate system to match data
CartesianCoordinates pTransform1 =
    new CartesianCoordinates( ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
.
.
.
// Add a custom mouse listener that calculates new averages whenever the data is
changed
MoveData mouselistener = new MoveData(this, pTransform1);
mouselistener.SetMarkerType( ChartObj.MARKER_CROSS);
mouselistener.SetMarkerSize(12);
mouselistener.SetMoveMode(ChartObj.MOVE_Y);
mouselistener.SetEnable(true);
chartVu.SetCurrentMouseListener(mouselistener);

```

[Visual Basic]

```

ChartView chartVu = Me;

```

```

' Create a 3D dataset
Dim Dataset1 As New ContourDataset("First", x1, y1, z1)
' Create a coordinate system to match data
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _
    ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
.
.
.
Dim mousetlistener As New MoveData(Me, pTransform1)
mousetlistener.SetMarkerType(ChartObj.MARKER_CROSS)
mousetlistener.SetMarkerSize(12)
mousetlistener.SetMoveMode(ChartObj.MOVE_Y)
mousetlistener.SetEnable(True)
chartVu.SetCurrentMouseListener(mousetlistener)

```


15. Zooming

ChartZoom

Zooming is the interactive re-scaling of a chart's physical coordinate system and the related axes based on limits defined by clicking and dragging a mouse inside the current graph window. A typical use of zooming is in applications where the initial chart displays a large number of data points. The user interacts with the chart, defining smaller and smaller zoom rectangles, zeroing in on the region of interest. The final chart displays axis limits that have a very small range compared to the range of the original, un-zoomed, chart. When a zoom rectangle is defined using a click and drag operation with the mouse, the drawing of the zoom rectangle is restricted to the XY plane. The zoom rectangle only changes the x- and y-scale of the graph. The z-scale of the chart remains unchanged.

Important zoom features include:

- Automatic recalculation of axis properties for tick mark spacing and axis labels.
- Zooming of time coordinates with smooth transitions between major scale changes: years->months->weeks->days->hours->minutes->seconds.
- Zooming of time coordinates that use a 5-day week and a non-24 hour day.
- Simultaneous zooming of an unlimited number of x- and y-coordinate systems and axes (super zooming).
- The user can recover previous zoom levels using a zoom stack.
- User-defineable zoom limits prevent numeric under and overflows

Simple Zooming of a single physical coordinate system

Class ChartZoom

GraphObj

|
+--ChartZoom

The **ChartZoom** class implements .Net delegates for mouse events. It implements and uses the mouse events: **OnMouseMove**, **OnMouseDown**, and **OnMouseUp**. The default operation of the **ChartZoom** class starts the zoom operation on the **OnMouseDown** event; it draws the zoom rectangle using the XOR drawing mode during the **OnMouseMove** event; and terminates the zoom operation on the **OnMouseUp** event.

During the **OnMouseUp** event, the zoom rectangle is converted from device units into the chart physical coordinates and this information is stored and optionally used to rescale the chart scale and all axis objects that reference the chart scale. If four axis objects reference a single chart scale, for example when axes bound a chart on all four sides, all four axes re-scale to match the new chart scale.

ChartZoom constructor

The constructor below creates a zoom object for a single chart coordinate system.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal component As ChartView, _
    ByVal transform As PhysicalCoordinates, _
    ByVal brescale As Boolean _
)

Overloads Public Sub New( _
    ByVal component As ChartView, _
    ByVal transform As PhysicalCoordinates, _

    ByVal zpos As Double
    ByVal brescale As Boolean _
)

[C#]
public ChartZoom(
    ChartView component,
    PhysicalCoordinates transform,
    bool brescale
);

public ChartZoom(
    ChartView component,
    PhysicalCoordinates transform,
    Double zvalue,
    bool brescale
);
```

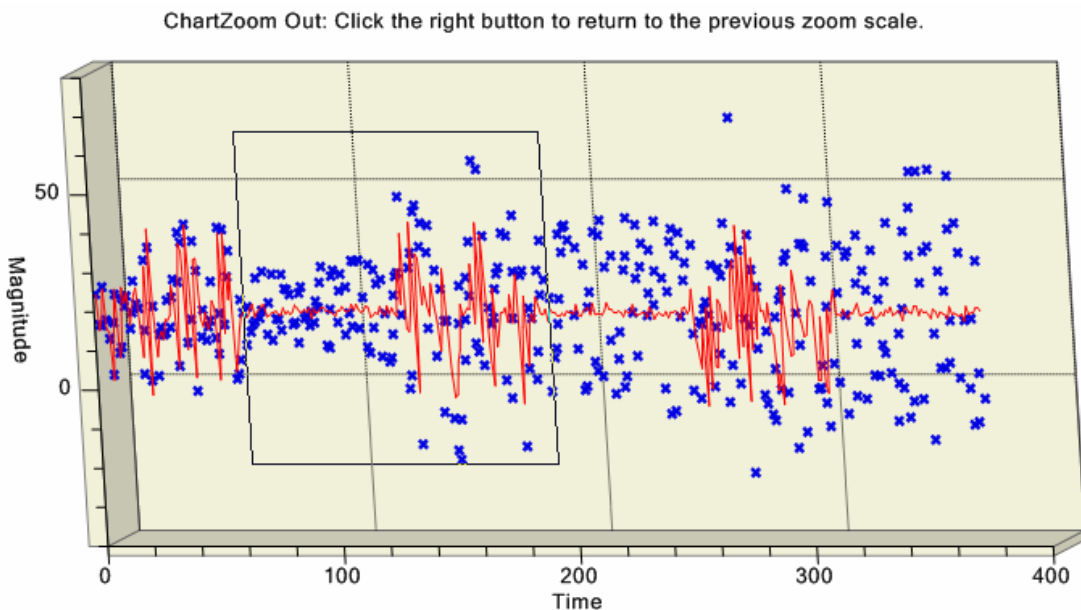
<i>component</i>	A reference to the ChartView object that the chart is placed in.
<i>transform</i>	The PhysicalCoordinates object associated with the scale being zoomed.
<i>zpos</i>	The z-scale position of the zoom rectangle.
<i>brescale</i>	True designates that the scale should be re-scaled, once the final zoom rectangle is ascertained.

Enable the zoom object after creation using the **ChartZoom.SetEnable(true)** method.

Retrieve the physical coordinates of the zoom rectangle using the **ChartZoom.GetZoomMin** and **GetZoomMax** methods. Restrict zooming in the x- or y-direction using the **SetZoomXEnable** and **SetZoomYEnable** methods. Set the rounding mode associated with rescale operations using the **SetZoomXRoundMode** and **SetZoomYRoundMode** methods. Call the **ChartZoom.PopZoomStack** method at any time and the chart scale reverts to the minimum and maximum values of the previous zoom operation. Repeated calls to the **PopZoomStack** method return the chart scale to its original condition, after which the **PopZoomStack** method has no effect.

Simple zoom example (extracted from the **ZoomExamples.SimpleZoom** example)

In this example, a new class derives from the **ChartZoom** class and the **OnMouseDown** event overridden. The event invokes the **PopZoomStack** method. Otherwise, the default operation of the **ChartZoom** class controls everything else.



Simple zooming (ZoomExamples.SimpleZoom)

[C#]

```
// Custom zoom class that implements a zoom stack
private class ZoomWithStack: ChartZoom
{
    public ZoomWithStack(ChartView component, CartesianCoordinates transform,
double zpos, bool brescale):
        base( component, transform, zpos, brescale)
```

340 *Zooming*

```
{
}

public override void OnMouseDown (MouseEventArgs mouseevent)
{
// if right mouse button, pop zoom stack one level
if ((mouseevent.Button & MouseButton.Right) != 0)
    this.PopZoomStack();
else
    base.OnMouseDown(mouseevent);
}
}
.
.
.
// Set up zoom mouse listener
double zpos = 0.0;
ZoomWithStack zoomObj = new ZoomWithStack(chartVu, pTransform1, zpos, true);
zoomObj.SetButtonMask(MouseButton.Left);
zoomObj.SetZoomYEnable(true);
zoomObj.SetZoomXEnable(true);
zoomObj.SetZoomXRoundMode(CharObj.AUTOAXES_FAR);
zoomObj.SetZoomYRoundMode(CharObj.AUTOAXES_FAR);
zoomObj.SetEnable(true);
zoomObj.SetZoomStackEnable(true);
// set range limits to 1000 ms, 1 degree
zoomObj.SetZoomRangeLimitsRatio(new Dimension3D(0.001, 0.001));
chartVu.SetCurrentMouseListener(zoomObj);
```

[Visual Basic]

```
' Custom zoom class that implements a zoom stack
Private Class ZoomWithStack Inherits ChartZoom
    Public Sub New(ByVal component As ChartView, _
        ByVal transform As CartesianCoordinates, _
        ByVal zpos As Double, ByVal brescale As Boolean)
        MyBase.New(component, transform, zpos, brescale)
    End Sub 'New

    Public Overrides Sub OnMouseDown(ByVal mouseevent As MouseEventArgs)
        ' if right mouse button, pop zoom stack one level
        If (mouseevent.Button And MouseButton.Right) <> 0 Then
            Me.PopZoomStack()
        Else
```



```

MyBase.OnMouseDown(mouseevent)

    End If
End Sub 'OnMouseDown

End Class 'ZoomWithStack

.
.
.

' Set up zoom mouse listener
Dim zpos As Double = 0.0
Dim zoomObj As New ZoomWithStack(chartVu, pTransform1, zpos, True)
zoomObj.SetButtonMask(MouseButtons.Left)
zoomObj.SetZoomYEnable(True)
zoomObj.SetZoomXEnable(True)
zoomObj.SetZoomXRoundMode(ChartObj.AUTOAXES_FAR)
zoomObj.SetZoomYRoundMode(ChartObj.AUTOAXES_FAR)
zoomObj.SetEnable(True)
zoomObj.SetZoomStackEnable(True)
' set range limits to 1000 ms, 1 degree
zoomObj.SetZoomRangeLimitsRatio(New Dimension3D(0.001, 0.001))
chartVu.SetCurrentMouseListener(zoomObj)

```

Super Zooming of multiple physical coordinate systems

The **ChartZoom** class also supports the zooming of multiple physical coordinate systems (*super zooming*). During the **OnMouseUp** event, the zoom rectangle is converted from device units into the physical coordinates of each scale, and this information is used to re-scale each coordinate system, and the axis objects associated with them.

Use the constructor below in order to super zoom a chart that has multiple coordinate systems and axes.

ChartZoom constructor

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal component As ChartView, _
    ByVal transforms As PhysicalCoordinates(), _
    ByVal brescale As Boolean _
)

Overloads Public Sub New( _
    ByVal component As ChartView, _
    ByVal transforms As PhysicalCoordinates(), _
    ByVal zpos As Double(), _

```

342 Zooming

```
        ByVal brescale As Boolean _
    )

[C#]
public ChartZoom(
    ChartView component,
    PhysicalCoordinates[] transforms,
    bool brescale
);

public ChartZoom(
    ChartView component,
    PhysicalCoordinates[] transforms,
    double zpos,
    bool brescale
);
```

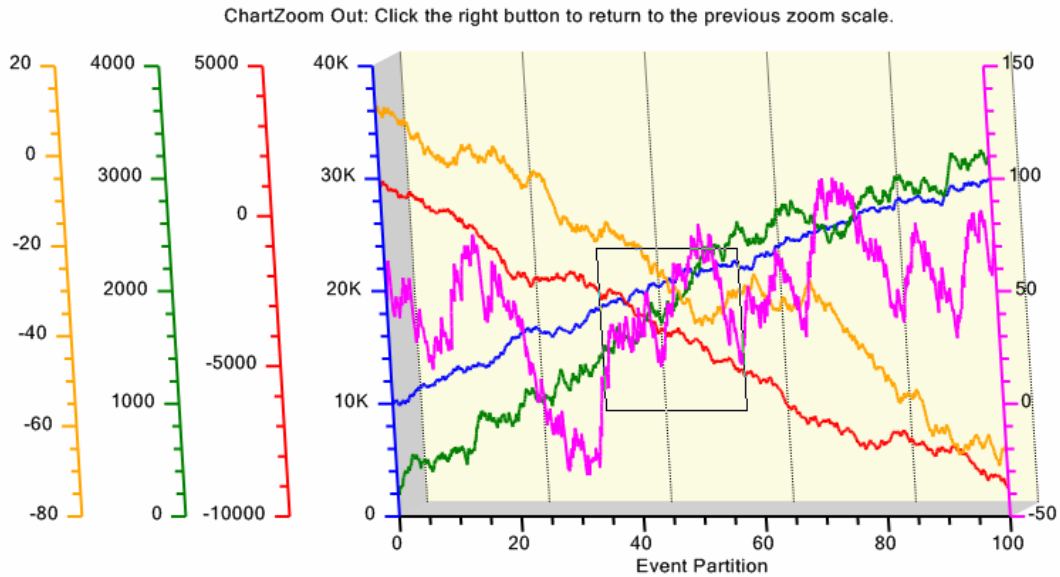
<i>component</i>	A reference to the ChartView object that the chart is placed in.
<i>transforms</i>	An array, size numtransforms, of the PhysicalCoordinates objects associated with the zoom operation.
<i>zpos</i>	The z-scale position of the zoom rectangle.
<i>brescale</i>	True designates that the all of the scales should be re-scaled, once the final zoom rectangle is ascertained.

Call the **ChartZoom.SetEnable(true)** method to enable the zoom object.

Restrict zooming in the x- or y-direction using the **SetZoomXEnable** and **SetZoomYEnable** methods. Set the rounding mode associated with rescale operations using the **SetZoomXRoundMode** and **SetZoomYRoundMode** methods. Call the **ChartZoom.PopZoomStack** method at any time and the chart scale reverts to the minimum and maximum values of the previous zoom operation. Repeated calls to the **PopZoomStack** method return the chart scale is to its original condition, after which the **PopZoomStack** method has no effect.

Super zoom example (Adapted from the **ZoomExamples.SuperZoom** example)

In this example, a new class derives from the **ChartZoom** class and the **OnMouseDown** event overridden. The event invokes the **PopZoomStack** method. Otherwise, the default operation of the **ChartZoom** class controls everything else.



Super zooming (ZoomExamples.SuperZoom)

[C#]

```
public class SuperZoom : com.quinncurtis.chart3dnet.ChartView
{
    // Custom zoom class that implements a zoom stack
    private class ZoomWithStack: ChartZoom
    {
        public ZoomWithStack(ChartView component, CartesianCoordinates []transforms,
            double zpos, bool brescale):
            base( component, transforms, zpos, brescale)
        {
        }
        public override void OnMouseDown (MouseEventArgs mouseevent)
        {
            // if right mouse button, pop zoom stack one level
            if ((mouseevent.Button & MouseButton.Right) != 0)
                this.PopZoomStack();
            else
                base.OnMouseDown(mouseevent);
        }
    }
}
```

344 *Zooming*

```
SimpleDataset Dataset1 = new SimpleDataset("First", x1,y1);
SimpleDataset Dataset2 = new SimpleDataset("Second",x1,y2);
SimpleDataset Dataset3 = new SimpleDataset("Third", x1,y3);
SimpleDataset Dataset4 = new SimpleDataset("Fourth",x1,y4);
SimpleDataset Dataset5 = new SimpleDataset("Fifth", x1,y5);

CartesianCoordinates pTransform1 =
    new CartesianCoordinates( ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);

CartesianCoordinates pTransform2 =
    new CartesianCoordinates( ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform2.AutoScale(Dataset2, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
CartesianCoordinates pTransform3 =
    new CartesianCoordinates( ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform3.AutoScale(Dataset3, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
CartesianCoordinates pTransform4 =
    new CartesianCoordinates( ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform4.AutoScale(Dataset4, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
CartesianCoordinates pTransform5 =
    new CartesianCoordinates( ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform5.AutoScale(Dataset5, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
.
.

// Define a zoom object using a custom zoom class
double zpos = 0.0;
CartesianCoordinates []transformArray = {pTransform1,
pTransform2,pTransform3,pTransform4,pTransform5};
ZoomWithStack zoomObj = new ZoomWithStack(chartVu, transformArray, zpos, true);
zoomObj.SetButtonMask(MouseButtons.Left);
zoomObj.SetZoomYEnable(true);
zoomObj.SetZoomXEnable(true);
zoomObj.SetZoomXRoundMode(ChartObj.AUTOAXES_FAR);
zoomObj.SetZoomYRoundMode(ChartObj.AUTOAXES_FAR);
zoomObj.SetEnable(true);
zoomObj.SetZoomStackEnable(true);
chartVu.SetCurrentMouseListener(zoomObj);
```

[Visual Basic]

```
Private Class ZoomWithStack Inherits ChartZoom
Public Sub New(ByVal component As ChartView, _
    ByVal transforms() As CartesianCoordinates, _
```

```

        ByVal n As Integer, ByVal brescale As Boolean)
        MyBase.New(component, transforms, brescale)
    End Sub 'New

    Public Overrides Sub OnMouseDown(ByVal mouseevent As MouseEventArgs)
        If (mouseevent.Button And System.Windows.Forms.MouseButtons.Right) <> 0 Then
            Me.PopZoomStack()
        Else
            MyBase.OnMouseDown(mouseevent)
        End If
    End Sub 'OnMouseDown
End Class 'ZoomWithStack

.
.
.
' Create 5 datasets
Dim Dataset1 As New SimpleDataset("First", x1, y1)
Dim Dataset2 As New SimpleDataset("Second", x1, y2)
Dim Dataset3 As New SimpleDataset("Third", x1, y3)
Dim Dataset4 As New SimpleDataset("Fourth", x1, y4)
Dim Dataset5 As New SimpleDataset("Fifth", x1, y5)

' Define 5 different coordinate systems, each uniquely scaled to a dataset
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _
    ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

Dim pTransform2 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _
    ChartObj.LINEAR_SCALE)
pTransform2.AutoScale(Dataset2, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

Dim pTransform3 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _
    ChartObj.LINEAR_SCALE)
pTransform3.AutoScale(Dataset3, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

Dim pTransform4 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _
    ChartObj.LINEAR_SCALE)
pTransform4.AutoScale(Dataset4, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

Dim pTransform5 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _
    ChartObj.LINEAR_SCALE)
pTransform5.AutoScale(Dataset5, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

```

```

.
.
.
' Define a zoom object using a custom zoom class
Dim zpos As Double = 0.0
Dim transformArray As CartesianCoordinates() = {pTransform1, pTransform2, _
    pTransform3, pTransform4, pTransform5}
Dim zoomObj As New ZoomWithStack(chartVu, transformArray, zpos, True)
zoomObj.SetButtonMask(MouseButtons.Left)
zoomObj.SetZoomYEnable(True)
zoomObj.SetZoomXEnable(True)
zoomObj.SetZoomXRoundMode(ChartObj.AUTOAXES_FAR)
zoomObj.SetZoomYRoundMode(ChartObj.AUTOAXES_FAR)
zoomObj.SetEnable(True)
zoomObj.SetZoomStackEnable(True)
chartVu.SetCurrentMouseListener(zoomObj)

```

Limiting the Zoom Range

A zoom window needs to have zoom limits placed on the minimum allowable zoom range for the x- and y-coordinates. Unrestricted, or infinite zooming can result in numeric under and overflows. The default minimum allowable range resulting from a zoom operation is 1/1000 of the original coordinate range. Change this value using the **ChartZoom.SetZoomRangeLimitsRatio** method. The minimum allowable range for this value is approximately 1.0e-9. Another way to set the minimum allowable range is to specify explicit values for the x- and y-range using the **ChartZoom.SetZoomRangeLimits** method. Specify the minimum allowable zoom range for a time axis in milliseconds, for example **ChartZoom.SetZoomRangeLimits(new Dimension(1000, 0.01))** sets the minimum zoom range for the time axis to 1 second and for the y-axis to 0.01. The utility method **ChartCalendar.GetCalendarWidthValue** is useful for calculating the milliseconds for any time base and any number of units. The code below sets a minimum zoom range of 45 minutes.

[C#]

```

double minZoomTimeRange =
    ChartCalendar.GetCalendarWidthValue(ChartObj.MINUTE, 45);
double minZoomYRange = 0.01;
Dimension zoomLimits = new Dimension(minZoomTimeRange, minZoomYRange);
zoomObj.SetZoomRangeLimits(zoomLimits);

```

[Visual Basic]

```
Dim minZoomTimeRange As double = _  
    ChartObj.GetCalendarWidthValue(ChartObj.MINUTE, 45)  
Dim minZoomYRange As Double = 0.01  
Dim zoomLimits As Dimension = New Dimension(minZoomTimeRange, minZoomYRange)  
zoomObj.SetZoomRangeLimits(zoomLimits)
```


16. Data Tooltips

DataToolTip

Tooltip is a catchall phrase for a popup window that displays useful information about an object. A data tooltip is a popup box that displays the value of a data point in a chart. The data value can consist of the x-value, the y-value, or both values for a given point in a chart. The tooltip values are displayed using the numeric and time formats supported by the **NumericLabel** and **TimeLabel** classes.

Simple Data Tooltips

Class DataToolTip

MouseListener

|
+-DataToolTip

The **DataToolTip** class implements .Net mouse event delegates. It implements and uses the mouse events: **OnMouseMove**, **OnMouseDown**, and **OnMouseUp**. The default operation of the **DataToolTip** class traps the mouse pressed event. It calculates which chart object intersects the mouse cursor and which data points for the intersecting object are closest. Next, it pops up a window and displays the x-value and/or y-value representing the data point. When the mouse button is released, the **OnMouseUp** event, the tooltip popup window is deleted.

The tooltip data point search algorithm is complicated by the situation that many chart objects occupy a much larger area than the data point that is represented. Bars are a good example. A bar can occupy a large area, yet the actual data value represented by the bar is only a small point at the top. The tooltip algorithm searches for an intersection of the bar and the mouse cursor, not an intersection of the data point and the mouse cursor. If a hit on the bar is detected, the data values represented by the bar are used as the values displayed in the tooltip. The tooltip symbol will highlight the actual data point at the top of the bar. The tooltip window will always popup at the cursor location, not the data point location. If you click anywhere on a bar, the tooltip window will popup at that location and display the data value represented by the top of the bar.

The tooltip data point search algorithm works with both simple and group data. When used with simple plot objects (**SimpleLinePlot**, **SimpleBarPlot**, etc.) it locates the xy data point associated with the mouse event. When used with group plot objects it locates

the x-value and the y-group value associated with the mouse event. It is able to differentiate between “stacked” group plot objects (**StackedBarPlot**, **StackedLinePlot**) and the other group plot objects that are not stacked (**GroupBarPlot**, **MultiLinePlot**, **OHLCPLOT**, **CandlestickPlot**, etc.). The tooltip values displayed in the tooltip window reflect the actual data values stored in the associated dataset and do not reflect the implicit summation that goes on in the display of stacked plot objects. You should not use symbols to highlight the tooltip data point for stacked objects since the position of the tooltip symbol in the chart will not take into account the stacked object summation.

DataToolTip constructors

The constructors below create a **DataToolTip** object.

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal component As ChartView _
)

Overloads Public Sub New( _
    ByVal component As ChartView, _
    ByVal buttonmask As MouseButtons _
)

[C#]
public DataToolTip(
    ChartView component
);

public DataToolTip(
    ChartView component,
    MouseButtons buttonmask
);
```

<i>component</i>	A reference to the ChartView object that the chart is placed in.
<i>buttonmask</i>	Specifies the mouse button that is trapped to invoke a move.

Create the **DataToolTip** object and then install it using the **ChartView.SetCurrentMouseListener** method. This adds the **DataToolTip** object as a **MouseListener** to the **ChartView** object. Enable/Disable the function using **DataToolTip.SetEnable** method. Call **ChartView.SetCurrentMouseListener(null)** to remove the object as a mouse listener for the chart view.

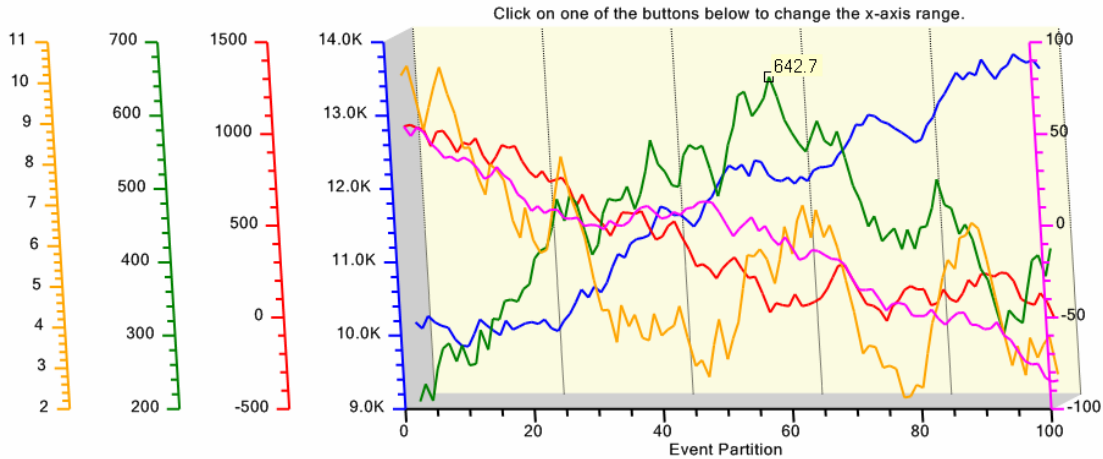
Set the threshold distance for deciding if the nearest data point found is a “hit” using the **DataToolTip.SetHitTestThreshold** method.

The default values for the **DataToolTip** class assume the following:

- The left mouse button pressed event is the trigger for the tooltip. Change this using the **DataToolTip.SetButtonMask** method.
- The numeric format for the x- and y-values are controlled by separate **NumericLabel** class templates that are both initially set to the **ChartObj.DECIMALFORMAT** format with a decimal precision of 1. Change this by creating a **NumericLabel** or **TimeLabel** object that specifies how you want the number formatted. Set the x- and y-value templates independently using the **DataToolTip.SetXValueTemplate** and **DataToolTip.SetYValue** template methods.
- The tooltip will display just the y-value of the selected object. Change this to display the x-value or both the x and y-values using the **DataToolTip.SetDataToolTipFormat** method, specifying one of the data tooltip format constants: **DATA_TOOLTIP_CUSTOM**, **DATA_TOOLTIP_X**, **DATA_TOOLTIP_Y**, **DATA_TOOLTIP_XY_ONELINE**, **DATA_TOOLTIP_TWOLINE**, **DATA_TOOLTIP_GROUP_MULTILINE**, **DATA_TOOLTIP_OHLC**.
- The tooltip popup window uses a default Sans Serif font with a size of 12. The text is justified above and to the right of the mouse cursor. The default background color of the data tooltip is a pale yellow, RGB (255,255,204). Change this by replacing the **ChartText** object used as the template for the tooltip popup window. Use the **DataToolTip.SetTextTemplate** method to replace the default template with your own.
- The selected data point is highlighted using a **ChartSymbol** object, set to a default shape of **ChartObj.SQUARE**, a size of 8 and a color of black. Change this by replacing the **ChartSymbol** used as the template with one of your own.

Simple data tooltip example (Extracted from the **MultipleAxes.MultiAxesChart example)**

In this example, the tooltip will display the decimal y-value of the nearest data point when the left mouse button is pressed.



Simple data tooltips (MultipleAxes.MultiAxesChart)

[C#]

```

ChartView chartVu;

DataToolTip datatooltip = new DataToolTip(chartVu);
chartVu.SetCurrentMouseListener(datatooltip);

```

[Visual Basic]

```

Dim chartVu As ChartView

Dim datatooltip As DataToolTip = New DataToolTip(chartVu)
chartVu.SetCurrentMouseListener(datatooltip)

```

Medium complex data tooltip example (Adapted from the SimpleLinePlots.LineFill example)

In this example, the tooltip will display the x-value of the data point as a date, and the y-value as currency. The x- and y-values are displayed on two separate lines, one above the other.

[C#]

```

Font tooltipFont = new Font("SansSerif", 10, FontStyle.Regular);

```

```

DataToolTip datatooltip = new DataToolTip(chartVu);
TimeLabel xValueTemplate = new TimeLabel( ChartObj.TIMEDATEFORMAT_MDY);
NumericLabel yValueTemplate = new NumericLabel( ChartObj.CURRENCYFORMAT,0);
datatooltip.GetToolTipSymbol().SetColor(Color.Green);
datatooltip.SetXValueTemplate(xValueTemplate);
datatooltip.SetYValueTemplate(yValueTemplate);
datatooltip.SetDataToolTipFormat(ChartObj.DATA_TOOLTIP_XY_TWOLINE);
datatooltip.SetEnable(true);
chartVu.SetCurrentMouseListener(datatooltip);

```

[Visual Basic]

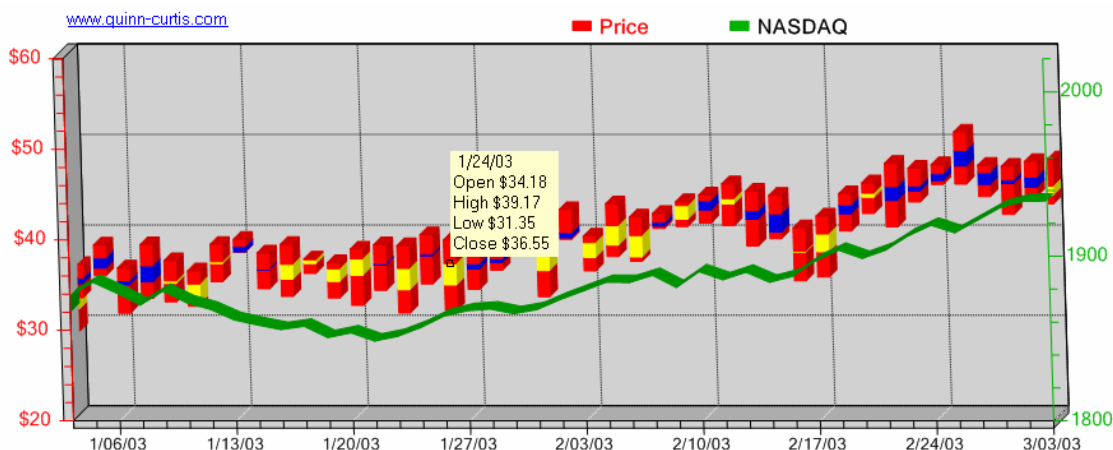
```

Dim toolTipFont As New Font("SansSerif", 10, FontStyle.Regular)
Dim datatooltip As New DataToolTip(chartVu)
Dim xValueTemplate As New TimeLabel(ChartObj.TIMEDATEFORMAT_MDY)
Dim yValueTemplate As New NumericLabel(ChartObj.CURRENCYFORMAT, 0)
datatooltip.GetToolTipSymbol().SetColor(Color.Green)
datatooltip.SetXValueTemplate(xValueTemplate)
datatooltip.SetYValueTemplate(yValueTemplate)
datatooltip.SetDataToolTipFormat(ChartObj.DATA_TOOLTIP_XY_TWOLINE)
datatooltip.SetEnable(True)
chartVu.SetCurrentMouseListener(datatooltip)

```

Complex data tooltip example (Adapted from the FinancialExamples.OpeningScreen example)

In this example, the tooltip will display the x-value of the data point as a date, and the y-value as currency. The x- and y-values are displayed on one line, side by side.



Complex data tooltip (FinancialExamples.OpeningScreen)

[C#]

```
// Define a data tooltip
Font toolTipFont = new Font("Microsoft Sans Serif", 10, FontStyle.Regular);
DataToolTip datatooltip = new DataToolTip(chartVu);
TimeLabel xValueTemplate = new TimeLabel( ChartObj.TIMEDATEFORMAT_MDY); // use
minimal constructor
NumericLabel yValueTemplate = new NumericLabel(ChartObj.CURRENCYFORMAT,2); // use
minimal constructor
ChartText textTemplate = new ChartText(toolTipFont,"");
textTemplate.SetTextBgColor(Color.FromArgb(255,255, 204));
textTemplate.SetTextBgMode(true);
ChartSymbol toolTipSymbol = new ChartSymbol(null, ChartObj.SQUARE, new
ChartAttribute(Color.Black));
toolTipSymbol.SetSymbolSize(5.0);
datatooltip.SetTextTemplate(textTemplate);
datatooltip.SetXValueTemplate(xValueTemplate);
datatooltip.SetYValueTemplate(yValueTemplate);
datatooltip.SetDataToolTipFormat(ChartObj.DATA_TOOLTIP_OHLC);
datatooltip.SetToolTipSymbol(toolTipSymbol);
datatooltip.SetEnable(true);
chartVu.SetCurrentMouseListener(datatooltip);
```

[Visual Basic]

```
Dim toolTipFont As New Font("SansSerif", 10, FontStyle.Regular)

Dim datatooltip As New DataToolTip(chartVu)
Dim xValueTemplate As New TimeLabel(ChartObj.TIMEDATEFORMAT_MDY)
Dim yValueTemplate As New NumericLabel(ChartObj.CURRENCYFORMAT, 2)
Dim textTemplate As New ChartText(toolTipFont, "")
textTemplate.SetTextBgColor(Color.FromArgb(255, 255, 204))
textTemplate.SetTextBgMode(True)
Dim toolTipSymbol As New ChartSymbol(Nothing, ChartObj.SQUARE, _
    New ChartAttribute(Color.Black))
toolTipSymbol.SetSymbolSize(5.0)
datatooltip.SetTextTemplate(textTemplate)
datatooltip.SetXValueTemplate(xValueTemplate)
datatooltip.SetYValueTemplate(yValueTemplate)
datatooltip.SetDataToolTipFormat(ChartObj.DATA_TOOLTIP_OHLC)
datatooltip.SetToolTipSymbol(toolTipSymbol)
datatooltip.SetEnable(True)
chartVu.SetCurrentMouseListener(datatooltip)
```

Custom Tooltip displays

It would be impossible to provide options for all possible tooltip displays. The **DataToolTip** class includes an option that enables the programmer to override the existing behavior of the class. The programmer is able to use the built in search routines to identify what plot object is selected, the dataset, the coordinate system and the actual data values associated with the plot object. Using this information the programmer can customize the text displayed in the **ChartText** object used to display the tooltip text.

Use the following steps to create a custom tooltip. See the FinancialExamples.OHLCChart for the complete example.

- Subclass the **CustomDataToolTip** class with one of your own.

```
class CustomToolTip: CustomDataToolTip
{
    ChartText stockpanel;
    .
    .
}
```

- Write your own constructor for the class, calling the base constructor.

```
public CustomToolTip(OHLCChart component): base (component)
{
    OHLCObj = component;
}
```

- Override the **Draw** event and build the string that you want displayed as the custom tooltip, and assign that string to the **CustomToolTipString** property .

```
override public void Draw(Graphics g2)
{
    DefineCustomerToolTipString();
    base.Draw(g2);
}

public void DefineCustomerToolTipString ()
{
    String tooltipstring = "";
    ChartPlot selectedPlot = (ChartPlot) GetSelectedPlotObj();
    if (selectedPlot != null)
    {
        .
        .
        .
    }
    // Assign as the tooltip string
```

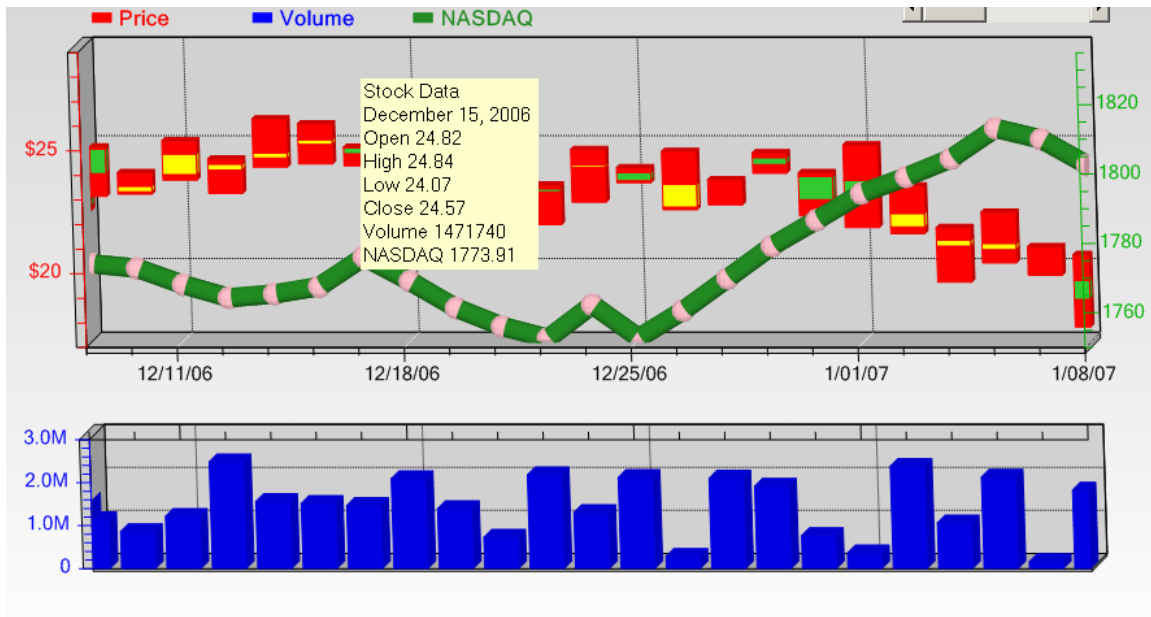
```

        this.CustomToolTipString = tooltipstring;
    }

```

Custom DataToolTip example (Extracted from the FinancialExamples.OHLCChart example)

In this example, a new class is derived from the **CustomDataToolTip** class and a custom tooltip string is defined when the overridden **Draw** method is called.



[C#]

```

ChartView chartVu;

class CustomToolTip: CustomDataToolTip
{
    OHLCChart OHLCObj = null;

    public CustomToolTip(OHLCChart component): base (component)
    {
        OHLCObj = component;
    }

    override public void Draw(Graphics g2)
    {
        DefineCustomerToolTipString();
        base.Draw(g2);
    }
}

```



```

}

public void DefineCustomerToolTipString ()
{
    String tooltipstring = "";
    ChartPlot selectedPlot = (ChartPlot) GetSelectedPlotObj();
    if (selectedPlot != null)
    {
        int selectedindex = GetNearestPoint().GetNearestPointIndex();
        PhysicalCoordinates transform = GetSelectedCoordinateSystem();

        tooltipstring = ("Stock Data" + "\n");
        // Looking to the original arrays, because we just have the selectedindex,
        // yet we want to display stock O-H-L-C data, volume and NASDAQ. Only one
        // of these datasets can be selected at a time by the tooltip.
        double open = OHLCObj.stockPriceData[0,selectedindex];
        double high = OHLCObj.stockPriceData[1,selectedindex];
        double low = OHLCObj.stockPriceData[2,selectedindex];
        double close = OHLCObj.stockPriceData[3,selectedindex];
        double nasdaq = OHLCObj.NASDAQData[selectedindex];
        double volume = OHLCObj.stockVolumeData[selectedindex];
        String openObj = ChartSupport.NumToString( open, _
            ChartObj.DECIMALFORMAT, 2, "");
        String highObj =ChartSupport.NumToString( high, _
            ChartObj.DECIMALFORMAT, 2, "");
        String lowObj = ChartSupport.NumToString( low, _
            ChartObj.DECIMALFORMAT, 2, "");
        String closeObj = ChartSupport.NumToString( close, _
            ChartObj.DECIMALFORMAT, 2, "");
        String volumeObj = ChartSupport.NumToString( volume, _
            ChartObj.DECIMALFORMAT, 0, "");
        String nasdaqObj = ChartSupport.NumToString( nasdaq, _
            ChartObj.DECIMALFORMAT, 2, "");
        TimeLabel timelabel = new TimeLabel(transform, _
            OHLCObj.xValues[selectedindex], ChartObj.TIMEDATEFORMAT_STANDARD);
        tooltipstring += (timelabel.GetTextString() + "\n");
        tooltipstring += ("Open " + openObj + "\n");
        tooltipstring += ("High " + highObj + "\n");
        tooltipstring += ("Low " + lowObj + "\n");
        tooltipstring += ("Close " + closeObj + "\n");
        tooltipstring += ("Volume " + volumeObj + "\n");
        tooltipstring += ("NASDAQ " + nasdaqObj);
    }
}

```

358 *Data Tooltips*

```
// Assign as the tooltip string
this.CustomToolTipString = tooltipstring;
}
}
}

CustomToolTip stocktooltip =
    new CustomToolTip(chartVu, xValues, stockPriceData, NASDAQData,
        stockVolumeData);
stocktooltip.SetDataToolTipFormat(ChartObj.DATA_TOOLTIP_CUSTOM);
stocktooltip.SetEnable(true);
chartVu.SetCurrentMouseListener(stocktooltip);
```

[Visual Basic]

```
Class CustomToolTip Inherits CustomDataToolTip
Private OHLCObj As OHLCChart = Nothing

Public Sub New(ByVal component As OHLCChart)
    MyBase.New(component)

    OHLCObj = component
End Sub 'New

Public Overrides Sub Draw(ByVal g2 As Graphics)
    DefineCustomerToolTipString()
    MyBase.Draw(g2)
End Sub 'Draw

Public Sub DefineCustomerToolTipString()
    Dim tooltipstring As String = ""

    Dim selectedPlot As ChartPlot = CType(GetSelectedPlotObj(), ChartPlot)
    If Not (selectedPlot Is Nothing) Then
        Dim selectedindex As Integer = GetNearestPoint().GetNearestPointIndex()
        Dim transform As PhysicalCoordinates = GetSelectedCoordinateSystem()

        tooltipstring = "Stock Data" + ControlChars.Lf
        ' Looking to the original arrays, because we just have the selectedindex,
```

```

' yet we want to display stock O-H-L-C data, volume and NASDAQ. Only one
' of these datasets can be selected at a time by the tooltip.
    Dim open As Double = OHLCObj.stockPriceData(0, selectedindex)
    Dim high As Double = OHLCObj.stockPriceData(1, selectedindex)
    Dim low As Double = OHLCObj.stockPriceData(2, selectedindex)
    Dim close As Double = OHLCObj.stockPriceData(3, selectedindex)
    Dim nasdaq As Double = OHLCObj.NASDAQData(selectedindex)
    Dim volume As Double = OHLCObj.stockVolumeData(selectedindex)
    Dim openObj As String = ChartSupport.NumToString(open, _
        ChartObj.DECIMALFORMAT, 2, "")
    Dim highObj As String = ChartSupport.NumToString(high, _
        _ChartObj.DECIMALFORMAT, 2, "")
    Dim lowObj As String = ChartSupport.NumToString(low, _
        ChartObj.DECIMALFORMAT, 2, "")
    Dim closeObj As String = ChartSupport.NumToString(close, _
        ChartObj.DECIMALFORMAT, 2, "")
    Dim volumeObj As String = ChartSupport.NumToString(volume, _
        ChartObj.DECIMALFORMAT, 0, "")
    Dim nasdaqObj As String = ChartSupport.NumToString(nasdaq, _
        ChartObj.DECIMALFORMAT, 2, "")
    Dim timelabel As New TimeLabel(transform, OHLCObj.xValues(selectedindex), _
        ChartObj.TIMEDATEFORMAT_STANDARD)

    tooltipstring += timelabel.GetTextString() + ControlChars.Lf
    tooltipstring += "Open " + openObj + ControlChars.Lf
    tooltipstring += "High " + highObj + ControlChars.Lf
    tooltipstring += "Low " + lowObj + ControlChars.Lf
    tooltipstring += "Close " + closeObj + ControlChars.Lf
    tooltipstring += "Volume " + volumeObj + ControlChars.Lf
    tooltipstring += "NASDAQ " + nasdaqObj

End If

' Assign as the tooltip string
Me.CustomToolTipString = tooltipstring
End Sub 'DefineCustomerToolTipString
End Class 'CustomToolTip

' Define custom tooltip
Dim stocktooltip As New CustomToolTip(Me)
stocktooltip.SetDataToolTipFormat(ChartObj.DATA_TOOLTIP_CUSTOM)
stocktooltip.SetEnable(True)
chartVu.SetCurrentMouseListener(stocktooltip)

```


17. Pie Charts

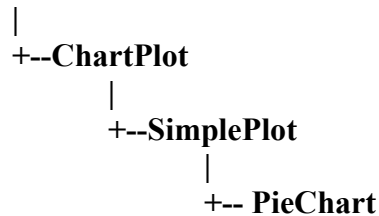
PieChart

Everyone is familiar with the ubiquitous pie chart. Pie charts are 1-dimensional, not because they are shallow, but because they represent a simple 1-dimensional series of numbers, {3, 5, 2, 7, 3, ...}, rather than the parametric set of data points { (3,2), (6,3), (7,3)...} used in the other plot types described in this software. The best use of pie charts involves data that has 10 or fewer elements. Otherwise, the text used to label the pie charts starts to overlap in adjacent, small pie wedges. The x-values of a dataset represent the data values for each pie wedge. The y-values of the dataset explode a pie wedge from its normal centered position. The position of the pie chart with respect to the z-axis is set using the attached datasets **ImplicitZValue** property. The datasets **ImplicitDepthValue** property specifies the 3D depth of pie chart.

Using the Pie Chart Class

Class PieChart

GraphObj



The **PieChart** class extends the **ChartPlot** class and displays pie charts. The x-values of the simple dataset used for data storage specify the pie wedge values. The y-values of the dataset specify the "explode" percentage for each pie wedge.

PieChartConstructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal dataset As SimpleDataset, _
    ByVal spiestring1s As String(), _
    ByVal attribs As ChartAttribute(), _
    ByVal labelinout1 As Integer, _
    ByVal pielabelformat As Integer _
)
```

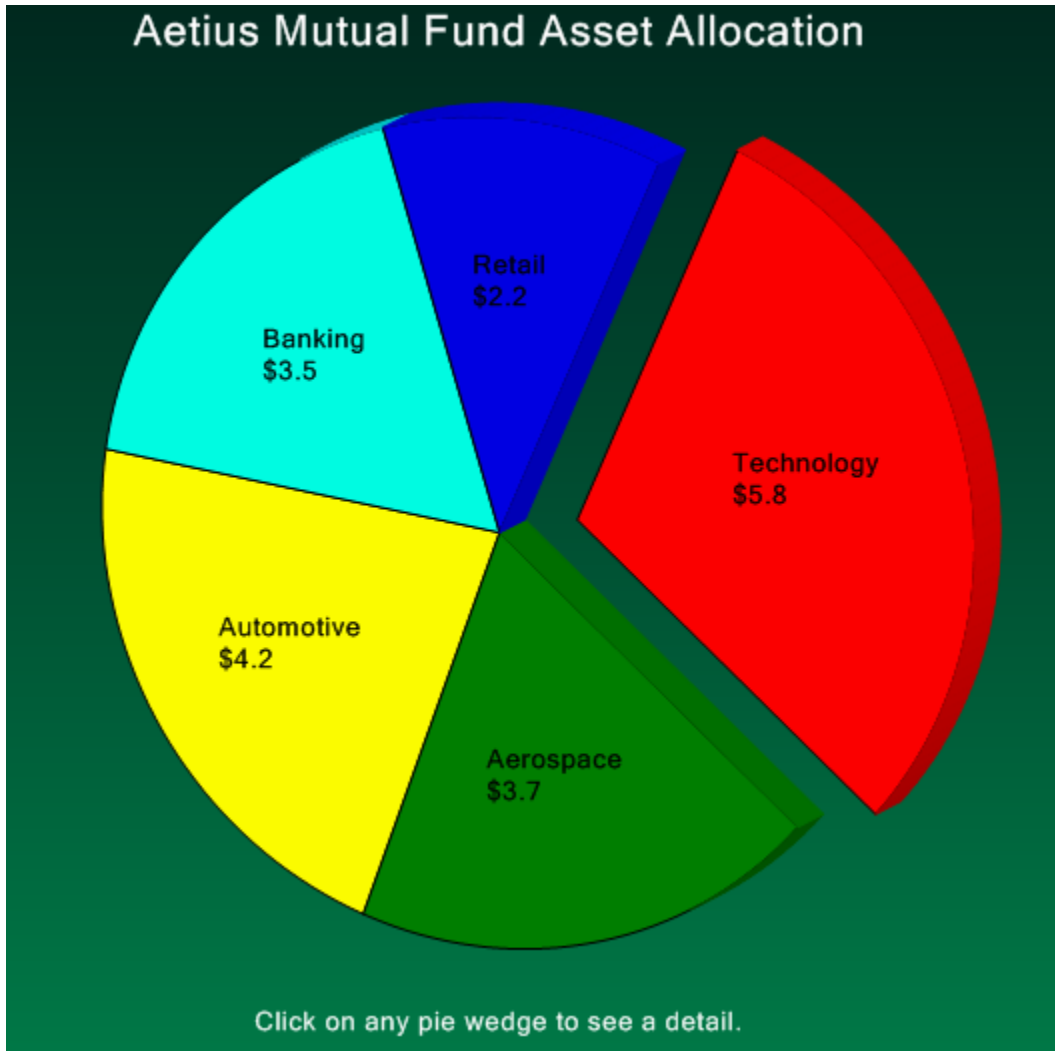
```
[C#]
public PieChart(
    PhysicalCoordinates transform,
    SimpleDataset dataset,
    string[] spiestrings,
    ChartAttribute[] attribs,
    int labelinout,
    int pielabelformat
);
```

<i>transform</i>	The pie chart is placed in the coordinate system defined by transform.
<i>dataset</i>	The pie chart represents the values in this dataset. The x-values of the simple dataset used for data storage specify the pie wedge values. The y-values of the dataset specify the "explode" percentage for each pie wedge.
<i>spiestrings</i>	An array of strings, size dataset. GetNumberDatapoints() , used as labels for the pie slices.
<i>attribs</i>	An array of ChartAttribute objects, size dataset. GetNumberDatapoints() that specify the attributes (outline color and fill color) for each wedge of a pie chart.
<i>labelinout</i>	An array of integer, size dataset. GetNumberDatapoints() , specifying if a specific pie slice text label is drawn inside the pie slice, or outside of the pie slice. Use one of the constants: PIELABEL_OUTSLICE or PIELABEL_INSLICE.
<i>pielabelformat</i>	All pie slice labels share the same format. Use one of the pie slice label format constants: PIELABEL_NONE Do not display and pie slice text PIELABEL_STRING Display only the pie text strings, no numeric values PIELABEL_NUMVALUE Display the pie numeric value only, no pie text strings. PIELABEL_STRINGNUMVAL Display the pie text string and numeric value.

A pie chart uses a default **CartesianCoordinates** object. Center it in the window using the **CartesianCoordinates.SetGraphBorderDiagonal** method. Format the text used to label the pie chart, both the strings and the numeric values, using a **NumericLabel** template set using the **PieChart.SetPlotLabelTemplate** method. Change the starting

position of the first pie wedge from the default value of 0.0 (3:00 position) using the **PieChart.SetStartPieSliceAngle** method.

Simple pie chart (extracted from the example program `PieCharts.SimplePieChart`)



Simple pie chart (`PieCharts.SimplePieChart`)

```

ChartView chartVu;
Font theFont;
theFont = new Font("Microsoft Sans Serif", 10, FontStyle.Bold);

String []labelArray;
chartVu = this;
// Define the pie segment labels
int numPoints = 5;

```

364 *Pie Charts*

```
String []sPieStrings = {"Technology", "Retail", "Banking", "Automotive",
"Aerospace"};

ChartAttribute []attribs = new ChartAttribute[5];

Color []colorArray = {Color.Red, Color.Blue, Color.Cyan, Color.Yellow,
Color.Green};

String techLabel;
String retailLabel;
String bankLabel;
String aeroLabel;
String autoLabel;

// create pie chart data
double []x1 = new double[numPoints];
double []y1 = new double[numPoints];
int i;

// x-values represent pie chart segment value
// y-values represent the explode value for cooresponding segment
x1[0] = 5.8; y1[0] = 0.2;
x1[1] = 2.2; y1[1] = 0.0;
x1[2] = 3.5; y1[2] = 0.0;
x1[3] = 4.2; y1[3] = 0.0;
x1[4] = 3.7; y1[4] = 0.0;
/ Define attribute for each pie segment, black outline with color fill
for (i=0; i < numPoints; i++)
{
    attribs[i] = new ChartAttribute (Color.Black, 1,DashStyle.Solid,
colorArray[i]);
}

// Define pie chart dataset
SimpleDataset Dataset1 = new SimpleDataset("First",x1,y1);
Dataset1.ImplicitDepthValue = 0.2;
Dataset1.ImplicitZValue = 0.4;

// Create a default cartesian coordinate system
CartesianCoordinates pTransform1 = new CartesianCoordinates();
// Position the pie inside the graph area
pTransform1.SetGraphBorderDiagonal(0.1, .1, .9, 0.9) ;

// Set rotational parameters
Point3D protate = new Point3D(10, 20, 0);
pTransform1.AbsRotateCoordinateSystem( protate);
```



```

// Define background
Background background1 = new Background( pTransform1, ChartObj.GRAPH_BACKGROUND,
    Color.FromArgb(0,120,70), Color.FromArgb(0,40,30), ChartObj.Y_AXIS);
chartVu.AddChartObject(background1);

// Define the pie chart
PieChart thePlot1 = new PieChart(pTransform1, Dataset1, sPieStrings,attribs,
    ChartObj.PIELABEL_OUTSLICE, ChartObj.PIELABEL_STRINGNUMVAL);
thePlot1.SetStartPieSliceAngle(-45);

// Define the numeric template for the pie chart segment values
NumericLabel labeltemplate = new NumericLabel();
labeltemplate.SetNumericFormat(ChartObj.CURRENCYFORMAT);
    labeltemplate.SetDecimalPos(1);
labeltemplate.SetTextFont(theFont);
thePlot1.SetPlotLabelTemplate(labeltemplate);
thePlot1.SetLabelInOut(0,ChartObj.PIELABEL_INSLICE);
thePlot1.SetLabelInOut(1,ChartObj.PIELABEL_INSLICE);
thePlot1.SetLabelInOut(2,ChartObj.PIELABEL_INSLICE);
thePlot1.SetLabelInOut(3,ChartObj.PIELABEL_INSLICE);
thePlot1.SetLabelInOut(4,ChartObj.PIELABEL_INSLICE);
chartVu.AddChartObject(thePlot1);

```

[Visual Basic]

```

Dim chartVu As ChartView
Dim theFont As Font
theFont = New Font("Microsoft Sans Serif", 10, FontStyle.Bold)

Dim labelArray() As String
chartVu = Me
' Define the pie segment labels
Dim numPoints As Integer = 5
Dim sPieStrings As String() = {"Technology", "Retail", "Banking", "Automotive",
    "Aerospace"}
Dim attribs(5) As ChartAttribute
Dim colorArray As Color() = {Color.Red, Color.Blue, Color.Cyan, Color.Yellow,
    Color.Green}

Dim techLabel As String

```

366 *Pie Charts*

```
Dim retailLabel As String
Dim bankLabel As String
Dim aeroLabel As String
Dim autoLabel As String

' create pie chart data
Dim x1(numPoints - 1) As Double
Dim y1(numPoints - 1) As Double
Dim i As Integer

' x-values represent pie chart segment value
' y-values represent the explode value for cooresponding segment
x1(0) = 5.8
y1(0) = 0.2
x1(1) = 2.2
y1(1) = 0.0
x1(2) = 3.5
y1(2) = 0.0
x1(3) = 4.2
y1(3) = 0.0
x1(4) = 3.7
y1(4) = 0.0

' Define attribute for each pie segment, black outline with color fill
For i = 0 To numPoints - 1
    attribs(i) = New ChartAttribute(Color.Black, 1, _
        DashStyle.Solid, colorArray(i))
Next i

' Define pie chart dataset
Dim Dataset1 As New SimpleDataset("First", x1, y1)
Dataset1.ImplicitDepthValue = 0.2
Dataset1.ImplicitZValue = 0.4

' Create a default cartesian coordinate system
Dim pTransform1 As New CartesianCoordinates()
' Position the pie inside the graph area
pTransform1.SetGraphBorderDiagonal(0.1, 0.1, 0.9, 0.9)

' Set rotational parameters
Dim protate As New Point3D(10, 20, 0)
pTransform1.AbsRotateCoordinateSystem(protate)
```

```

' Define background
Dim background1 As New Background(pTransform1, _
    ChartObj.GRAPH_BACKGROUND, Color.FromArgb(0, 120, 70), _
    Color.FromArgb(0, 40, 30), ChartObj.Y_AXIS)
chartVu.AddChartObject(background1)

' Define the pie chart
Dim thePlot1 As New PieChart(pTransform1, Dataset1, sPieStrings, _
    attribs, ChartObj.PIELABEL_OUTSLICE, ChartObj.PIELABEL_STRINGNUMVAL)
thePlot1.SetStartPieSliceAngle(-45)

' Define the numeric template for the pie chart segment values
Dim labeltemplate As New NumericLabel()
labeltemplate.SetNumericFormat(ChartObj.CURRENCYFORMAT)
labeltemplate.SetDecimalPos(1)
labeltemplate.SetTextFont(theFont)
thePlot1.SetPlotLabelTemplate(labeltemplate)
thePlot1.SetLabelInOut(0, ChartObj.PIELABEL_INSLICE)
thePlot1.SetLabelInOut(1, ChartObj.PIELABEL_INSLICE)
thePlot1.SetLabelInOut(2, ChartObj.PIELABEL_INSLICE)
thePlot1.SetLabelInOut(3, ChartObj.PIELABEL_INSLICE)
thePlot1.SetLabelInOut(4, ChartObj.PIELABEL_INSLICE)
chartVu.AddChartObject(thePlot1)

```


19. Legends

Legend

StandardLegend

BubblePlotLegend

Charts containing multiple chart objects, line plots, bar graphs and scatter plots for example, usually require a legend. The legend provides a key so that the viewer of the chart can figure out what data is associated with what chart object. The bounding box of the legend is rectangular and can reside anywhere in the chart window: inside the plot area volume, overlapping it or completely outside. The legend rectangle can have a border and can be filled with a solid color or left transparent. The legend object can hold one or more legend items, where each legend item is a symbol-text string combination providing the key for one of the plot objects in the graph. The legend can also have a title and footer.

The **Legend** class is the abstract base class for chart legends. It organizes a collection of legend items as a rectangular object. Legends are 2-dimensional. They are positioned in the overall rectangle of the **ChartView** viewport and do not rotate with the chart.

The **StandardLegend** is a subclass of the **Legend** class and it is the primary legend class for all plot objects except for bubble plots. The legend items objects display in a row or column format. Each legend item contains a symbol and descriptive string. The symbol normally associates the legend item to a particular plot object, and the descriptive string describes what the plot object represents.

The **BubblePlotLegend** is a subclass of the **Legend** class and it is the legend class for bubble plots. The legend items objects display as offset, concentric circles with descriptive text giving the key for the value associated with a bubble of this size.

Standard Legends

Class StandardLegend

GraphObj

|

+-- **StandardLegend**

The **StandardLegend** is the primary legend class for all plot objects except for bubble plots. The class manages a list of **LegendItems** that holds the symbols and descriptive text for the symbols.

StandardLegend constructors

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal rx As Double, _
    ByVal ry As Double, _
    ByVal attrib As ChartAttribute, _
    ByVal nlayoutmode As Integer _
)

Overloads Public Sub New( _
    ByVal rx As Double, _
    ByVal ry As Double, _
    ByVal rwidth As Double, _
    ByVal rheight As Double, _
    ByVal attrib As ChartAttribute, _
    ByVal nlayoutmode As Integer _
)

[C#]
public StandardLegend(
    double rx,
    double ry,
    ChartAttribute attrib,
    int nlayoutmode
);

public StandardLegend(
    double rx,
    double ry,
    double rwidth,
    double rheight,
    ChartAttribute attrib,
    int nlayoutmode
);

```

<i>rx</i>	The x-position, in chart normalized coordinates, of the legend rectangle.
<i>ry</i>	The y-position, in chart normalized coordinates, of the legend rectangle.
<i>rwidth</i>	The width, in chart normalized coordinates, of the legend rectangle.
<i>rheight</i>	The height, in chart normalized coordinates, of the legend rectangle.
<i>attrib</i>	Specifies the outline color, outline line style, and fill color for the legend rectangle.
<i>nlayoutmode</i>	Specifies if the legend has a horizontal, or vertical layout. Use one of the orientation constants: <code>HORIZ_DIR</code> (row major) or <code>VERT_DIR</code> (column major).

Add legend items to a legend using one of the **AddLegendItem** methods.

AddLegendItem methods

[Visual Basic]

```
Overloads Public Function AddLegendItem( _
    ByVal stext As String, _
    ByVal nsymbol As Integer, _
    ByVal attrib As ChartAttribute, _
    ByVal thefont As Font _
) As Integer
```

```
Overloads Public Function AddLegendItem( _
    ByVal stext As String, _
    ByVal symbolshape As GraphicsPath, _
    ByVal attrib As ChartAttribute, _
    ByVal thefont As Font _
) As Integer
```

```
Overloads Public Function AddLegendItem( _
    ByVal legenditem As LegendItem _
) As Integer
```

```
Overloads Public Function AddLegendItem( _
    ByVal stext As String, _
    ByVal nsymbol As Integer, _
    ByVal chartobj As GraphObj, _
    ByVal thefont As Font _
) As Integer
```

```
Overloads Public Function AddLegendItem( _
    ByVal stext As String, _
    ByVal symbolshape As GraphicsPath, _
    ByVal chartobj As GraphObj, _
    ByVal thefont As Font _
) As Integer
```

```
Overloads Public Function AddLegendItem( _
    ByVal stext As String, _
    ByVal nsymbol As Integer, _
    ByVal chartobj As ChartPlot, _
    ByVal ngroup As Integer, _
    ByVal thefont As Font _
) As Integer
```

[C#]

```
public int AddLegendItem(
    string stext,
    int nsymbol,
    ChartAttribute attrib,
    Font thefont
);
```

```
public int AddLegendItem(
    string stext,
    GraphicsPath symbolshape,
    ChartAttribute attrib,
    Font thefont
);
```

```
public int AddLegendItem(
    LegendItem legenditem
);
```

```
public int AddLegendItem(
    string stext,
    int nsymbol,
    GraphObj chartobj,
    Font thefont
);
```

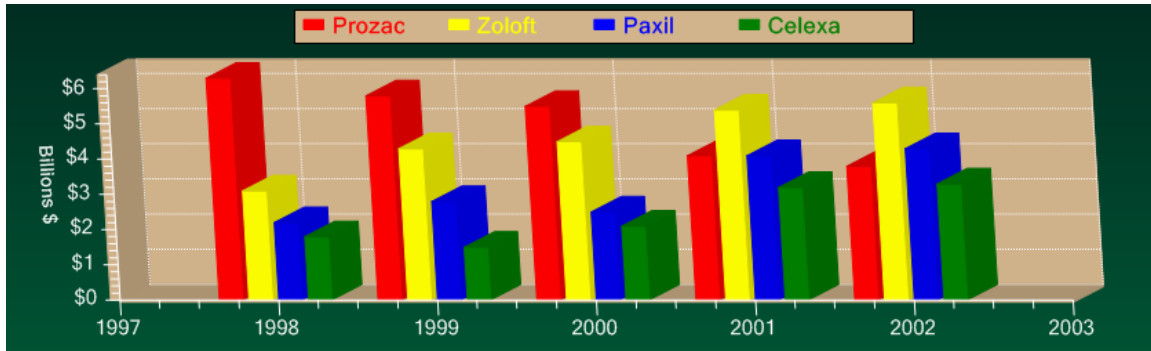
```
public int AddLegendItem(
    string stext,
    GraphicsPath symbolshape,
    GraphObj chartobj,
    Font thefont
);
```

```
public int AddLegendItem(
    string stext,
    int nsymbol,
    ChartPlot chartobj,
    int ngroup,
    Font thefont
);
```

<i>stext</i>	Specifies the text string for the legend item.
<i>nsymbol</i>	Specifies the symbol for the legend item. Use one of the chart symbol constants: 2D Symbols : NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE. 3D Symbols : HBAR3D, VBAR3D, CUBE3D, PLUS3D, DIAMOND3D, PYRAMID3D, HOURGLASS3D, ARROW3D, CYLINDER3D, SPHERE3D.
<i>chartobj</i>	The color and fill attributes for the legend item are copied from the attributes of this ChartPlot object.
<i>symbolshape</i>	Specifies a user defined shape to use as the legend item symbol.
<i>attrib</i>	Specifies the ChartAttribute object to get the color and fill attributes of the legend item.
<i>thefont</i>	Specifies the text font for the legend item.

The AddLegendItem returns the current number of legend items.

Simple legend example (extracted from the example program Bargraphs.GroupBargraphs)



Group bar plot with legend (Bargraphs.GroupBargraphs)

[C#]

```
.
.
.
.
// Define the legend
Font legendFont = new Font("Microsoft Sans Serif", 12, FontStyle.Bold);
ChartAttribute legendAttributes = new ChartAttribute(Color.Black, 1,
DashStyle.Solid, Color.Tan);
legendAttributes.SetLineFlag(true);
StandardLegend legend = new StandardLegend(0.25, 0.08, 0.6, 0.2,
    legendAttributes, StandardLegend.HORIZ_DIR);
legend.AddLegendItem("Prozac", ChartObj.HBAR, attrib1, legendFont);
legend.AddLegendItem("Zoloft", ChartObj.HBAR, attrib2, legendFont);
legend.AddLegendItem("Paxil", ChartObj.HBAR, attrib3, legendFont);
legend.AddLegendItem("Celexa", ChartObj.HBAR, attrib4, legendFont);
```

[VB]

```
' Define the legend
Dim legendFont As New Font("Microsoft Sans Serif", 12, FontStyle.Bold)
Dim legendAttributes As New ChartAttribute(Color.Black, 1, DashStyle.Solid,
Color.Tan)
legendAttributes.SetLineFlag(True)
Dim legend As New StandardLegend(0.25, 0.08, 0.6, 0.2, legendAttributes,
StandardLegend.HORIZ_DIR)
```

```

legend.AddLegendItem("Prozac", ChartObj.HBAR, attrib1, legendFont)
legend.AddLegendItem("Zoloft", ChartObj.HBAR, attrib2, legendFont)
legend.AddLegendItem("Paxil", ChartObj.HBAR, attrib3, legendFont)
legend.AddLegendItem("Celexa", ChartObj.HBAR, attrib4, legendFont)

```

Bubble Plot Legends

Class BubblePlotLegend

GraphObj

```

|
+-- BubblePlotLegend

```

The **BubblePlotLegend** is the primary legend class for bubble plots. The class manages a list of **BubblePlotLegendItem** that holds the symbols and descriptive text for the symbols.

BubblePlotLegend constructors

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal plot As BubblePlot, _
    ByVal rx As Double, _
    ByVal ry As Double, _
    ByVal rwidth As Double, _
    ByVal rheight As Double, _
    ByVal attrib As ChartAttribute _
)

```

```

Overloads Public Sub New( _
    ByVal plot As BubblePlot, _
    ByVal rx As Double, _
    ByVal ry As Double, _
    ByVal attrib As ChartAttribute _
)

```

```

[C#]
public BubblePlotLegend(
    BubblePlot plot,
    double rx,
    double ry,
    double rwidth,
    double rheight,
    ChartAttribute attrib
);

```

```

public BubblePlotLegend(
    BubblePlot plot,
    double rx,
    double ry,
    ChartAttribute attrib
);

```

<i>plot</i>	The bubble plot object the legend is associated with.
<i>rx</i>	The x-position, in chart normalized coordinates, of the legend rectangle.
<i>ry</i>	The y-position, in chart normalized coordinates, of the legend rectangle.
<i>rwidth</i>	The width, in chart normalized coordinates, of the legend rectangle.
<i>rheight</i>	The height, in chart normalized coordinates, of the legend rectangle.
<i>attrib</i>	Specifies the outline color, outline line style, and fill color for the legend rectangle.

Add legend items to a legend using one of the AddLegendItem methods.

AddLegendItem methods

```

[Visual Basic]
Overloads Public Function AddLegendItem( _
    ByVal stext As String, _
    ByVal rsize As Double, _
    ByVal attrib As ChartAttribute, _
    ByVal thefont As Font _
) As Integer

```

```

Overloads Public Sub New( _
    ByVal plot As BubblePlot, _
    ByVal rx As Double, _
    ByVal ry As Double, _
    ByVal rwidth As Double, _
    ByVal rheight As Double, _
    ByVal attrib As ChartAttribute _
)

```

```

[C#]
public int AddLegendItem(
    string stext,
    double rsize,
    ChartAttribute attrib,
    Font thefont
);

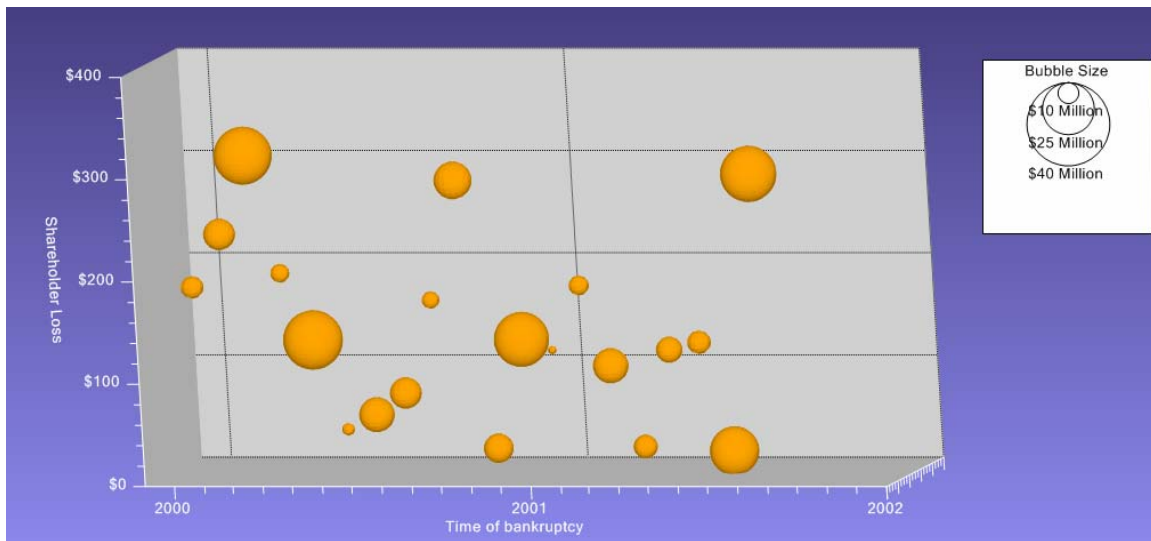
```

```
public BubblePlotLegend(
    BubblePlot plot,
    double rx,
    double ry,
    double rwidth,
    double rheight,
    ChartAttribute attrib
);
```

- stext* Specifies the text string for the legend item.
- rsiz* Specifies the size of the bubble for this item, in the same units as the coordinate system the bubble plot is placed in.
- chartobj* The color and fill attributes for the legend item are copied from the attributes of this chart object.
- thefont* Specifies the text font for the legend item.

The method returns the current number of legend items.

Simple legend example (extracted from the example program ScatterPlots.BubbleChart)



Bubble plot: sphere mode. (ScatterPlots.BubbleChart)

[C#]

```

.
.
.
// Define a bubble plot
ChartAttribute attrib1 = new ChartAttribute (Color.Orange, 0,DashStyle.Solid,
Color.Orange);

BubblePlot thePlot1 = new BubblePlot(pTransform1, Dataset1,
ChartObj.SIZE_BUBBLE_RADIUS, attrib1);

thePlot1.ImplicitDepthValue = 0.05;
thePlot1.BubblePlotMode = ChartObj.SPHERE3D;
chartVu.AddChartObject(thePlot1);

.
.

// Create a generic legend
ChartAttribute attrib2 = new ChartAttribute (Color.Black, 0,DashStyle.Solid);
attrib1.SetFillColor (Color.FromArgb(177, 33, 33));
Font legendFont = new Font("Microsoft Sans Serif", 10, FontStyle.Regular);
ChartAttribute legendAttributes = new ChartAttribute(Color.Black,
1,DashStyle.Solid, Color.White);
legendAttributes.SetFillFlag(true);
legendAttributes.SetLineFlag(true);

// Create bubble plot legend
BubblePlotLegend legend = new BubblePlotLegend(thePlot1, 0.85, 0.15, 0.14, 0.25,
legendAttributes);

legend.AddLegendItem("$10 Million",10, attrib2, legendFont);
legend.AddLegendItem("$25 Million", 25, attrib2, legendFont);
legend.AddLegendItem("$40 Million", 40, attrib2, legendFont);
legend.AddLegendGeneralText(ChartObj.LEGEND_HEADER, "Bubble Size", Color.Black,
legendFont);
chartVu.AddChartObject(legend);

```

[Visual Basic]

```

' Define a bubble plot
Dim attrib1 As New ChartAttribute(Color.Orange, 0, DashStyle.Solid, Color.Orange)
Dim thePlot1 As New BubblePlot(pTransform1, Dataset1, ChartObj.SIZE_BUBBLE_RADIUS,
attrib1)

thePlot1.ImplicitDepthValue = 0.05
thePlot1.BubblePlotMode = ChartObj.SPHERE3D
chartVu.AddChartObject(thePlot1)

.

```

380 *Legends*

```
.  
.  
' Create a generic legend  
Dim attrib2 As New ChartAttribute(Color.Black, 0, DashStyle.Solid)  
attrib1.SetFillColor(Color.FromArgb(177, 33, 33))  
Dim legendFont As New Font("Microsoft Sans Serif", 10, FontStyle.Regular)  
Dim legendAttributes As New ChartAttribute(Color.Black, 1, DashStyle.Solid,  
Color.White)  
legendAttributes.SetFillFlag(True)  
legendAttributes.SetLineFlag(True)  
  
' Create bubble plot legend  
Dim legend As New BubblePlotLegend(thePlot1, 0.85, 0.15, 0.14, 0.25,  
legendAttributes)  
legend.AddLegendItem("$10 Million", 10, attrib2, legendFont)  
legend.AddLegendItem("$25 Million", 25, attrib2, legendFont)  
legend.AddLegendItem("$40 Million", 40, attrib2, legendFont)  
legend.AddLegendGeneralText(ChartObj.LEGEND_HEADER, "Bubble Size", Color.Black,  
legendFont)  
chartVu.AddChartObject(legend)
```


20. Text Classes

ChartText

ChartTitle

AxisTitle

ChartLabel

StringLabel

TimeLabel

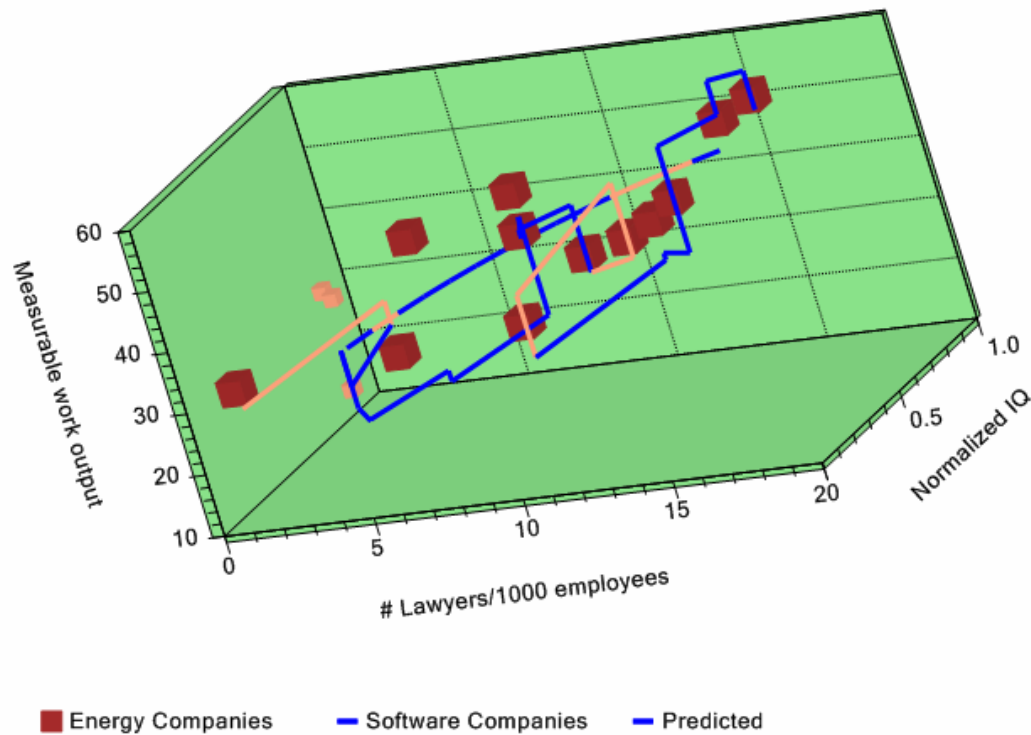
NumericLabel

The software uses the **ChartText** classes to position and format text in a chart. Examples of classes derived from the **ChartText** include the **ChartLabel**, **AxisLabels**, **ChartTitle**, and **AxisTitle** classes. The **Legend**, **PieChart** and **ChartPlot** classes, while not derived from the text classes, use them internally. **ChartText** derived classes are positioned in 3D space, the position of the text with respect to the z-axis is set using the text objects **ImplicitZValue** property.

Important note about ChartText rotations

You will find that sometimes you want to position text that is dependent on the chart coordinate system, including the 3D rotational transforms that the coordinate system goes through before it is rendered to the screen. Examples are chart annotations that must maintain a fixed relationship to physical coordinates of the charts plotting area. In this example you use a constructor that has as a parameter the coordinate system that you want the text to rotate with. Other times you will want the opposite to be true; even though the chart can be rotated, you want certain text to remain fixed with respect to the screen viewport of the **ChartView** window. Examples of this are the chart titles, footers and legends. All of the **ChartText** derived classes have versions that do NOT require that you pass in a coordinate system as the first parameter. In this case, a default, Cartesian coordinate system is created for that object, and that coordinate system is NOT rotated when the other coordinate systems of the chart are rotated. You can position the text using either a plot area normalized (NORM_PLOT_POS) , or graph area normalized (NORM_GRAPH_POS) coordinate system.

When **ChartText** is attached to a charts coordinate system, the text is positioned in the 3D space of that coordinate system. As the 3D chart is rotate, the *starting position* of the text will rotate with it. The rendered text string does NOT undergo a total 3D transformation however. This because all text is rendered by the .Net System.Drawing and .Net Drawing2D classes, which use the underlying TrueType and Postscript font engines for rendering. These rendering engines only support 2D rendering, with simple 2D translation, scaling and rotation, and not 3D transformations. The **ChartText** class does some clever calculations and captures the apparent angle of the projection of the 3D text object onto the 2D screen image, and rotates the .Net text accordingly in 2D space. The result is that when you rotate the chart about one or more axes, the text will rotate with it, maintaing a parallel or orthogonal angle to the axes.



Line plots and Scatter plots can be used to display 2D and 3D data.

A rotated version of the SimpleLinePlots.LinePlot3D example program

In the example above, note how the axes numeric labels, and the axes titles, rotate with the associated axis. The legend and the footer do not rotate with the chart, because they are not attached to the rotated charts coordinate system. The text does not appear to rotate into the z-axis plane though, and does not appear “forshortened” as you will see in some software packages (OpenGL for instance) that support full 3D text rendering. 3D “forshortened” text, while a good visual clue for depth, quickly becomes unreadable as the 3D text is rotated into the z-axis of the screen by more than 30 degrees. Also, true 3D rendering of text must be handled outside of the .Net TrueType and PostScript rendering engines and results in poor quality text, and slow rendering speeds on both the screen and the printer.

Simple Text Classes

Class ChartText

GraphObj

```

|
+--ChartText

```

The **ChartText** class is the base class for all text output classes. The **ChartText** class formats and places text in a chart. Position the **ChartText** objects using any of the coordinate systems. Rotate and justify the text vertically and horizontally. Insert a CR (carriage return, ASCII 13) character at line breaks for multiline text. The most common constructors are:

ChartText constructors

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal tfont As Font, _
    ByVal tstring As String, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal npostype As Integer, _
    ByVal xjust As Integer, _
    ByVal yjust As Integer, _
    ByVal rotation As Integer _
)

Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal tfont As Font, _
    ByVal tstring As String, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal npostype As Integer _
)

Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal tfont As Font, _
    ByVal tstring As String, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal z As Double, _
    ByVal npostype As Integer, _
    ByVal xjust As Integer, _
    ByVal yjust As Integer, _
    ByVal rotation As Integer _
)

C#
public ChartText(
    PhysicalCoordinates transform,
    Font tfont,
    string tstring,
    double x,
    double y,
    int npostype,
    int xjust,
    int yjust,
    int rotation
);

public ChartText(
    PhysicalCoordinates transform,
    Font tfont,
    string tstring,
    double x,
    double y,
    int npostype
);
```

```

public ChartText(
    PhysicalCoordinates transform,
    Font tfont,
    string tstring,
    double x,
    double y,
    double z,
    int npostype,
    int xjust,
    int yjust,
    int rotation
);

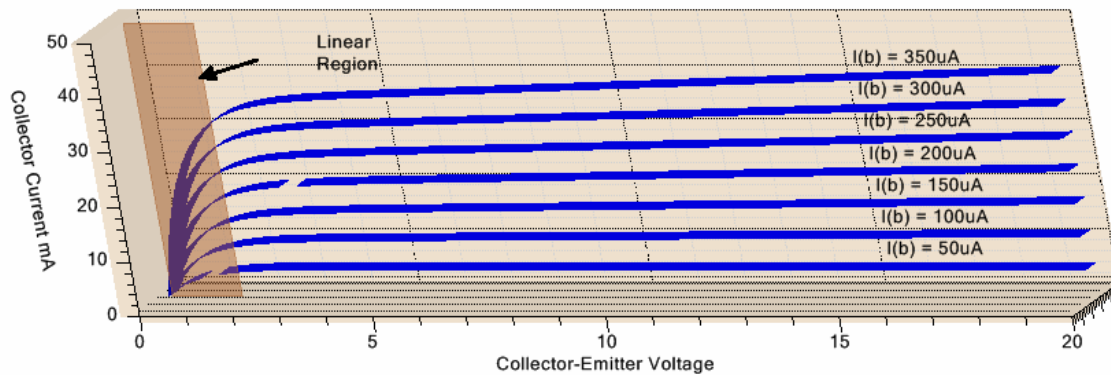
```

<i>transform</i>	Places the text in the coordinate system defined by transform.
<i>tfont</i>	A reference to a Font object.
<i>tstring</i>	A reference to a string object.
<i>x</i>	Specifies the x-value of the text position
<i>y</i>	Specifies the y-value of the text position
<i>z</i>	Specifies the z-value of the text position
<i>npostype</i>	Specifies the if the position of the text is specified in physical coordinates, normalized coordinates or .Net device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.
<i>xjust</i>	Specifies the horizontal justification of the text. Use one of the text justification constants: JUSTIFY_MIN, JUSTIFY_CENTER or JUSTIFY_MAX.
<i>yjust</i>	Specifies the vertical justification of the text. Use one of the text justification constants: JUSTIFY_MIN, JUSTIFY_CENTER or JUSTIFY_MAX.
<i>rotation</i>	The rotation (-360 to 360 degrees) of the text in the normal viewing plane.

Place text in is a time coordinate system (**TimeCoordinates**) by converting the time x-position to milliseconds and using the milliseconds as the x-position value.

See the special note, ***Special note about ChartText rotations*** at the beginning of this chapter.

ChartText example (extracted from the example program MultiLinePlots.Multilines)



Multi-ribbon plots (MultiLinePlots.MultiLines1)

[C#]

```
// Create a transparent rectangle shape object and position it in 3D space
double zdepth = Dataset1.ImplicitZValue;;

.
.
.

// Place an annotation at each line plot
Font theLabelFont = new Font("Microsoft Sans Serif", 10, FontStyle.Bold);
ChartText currentLabel1 = new ChartText(pTransform1, theLabelFont, "I(b) = 50uA",
    15.5, y1[0,85]+1,zdepth, ChartObj.PHYS_POS);
chartVu.AddChartObject(currentLabel1);

ChartText currentLabel2 = new ChartText(pTransform1, theLabelFont, "I(b) = 100uA",
    15.5, y1[1,85]+1,zdepth, ChartObj.PHYS_POS);
chartVu.AddChartObject(currentLabel2);

ChartText currentLabel3 = new ChartText(pTransform1, theLabelFont, "I(b) = 150uA",
    15.5, y1[2,85]+1,zdepth, ChartObj.PHYS_POS);
chartVu.AddChartObject(currentLabel3);

ChartText currentLabel4 = new ChartText(pTransform1, theLabelFont, "I(b) = 200uA",
    15.5, y1[3,85]+1,zdepth, ChartObj.PHYS_POS);
chartVu.AddChartObject(currentLabel4);

ChartText currentLabel5 = new ChartText(pTransform1, theLabelFont, "I(b) = 250uA",
    15.5, y1[4,85]+1,zdepth, ChartObj.PHYS_POS);
chartVu.AddChartObject(currentLabel5);

ChartText currentLabel6 = new ChartText(pTransform1, theLabelFont, "I(b) = 300uA",
    15.5, y1[5,85]+1,zdepth, ChartObj.PHYS_POS);
```

386 Text Classes

```
chartVu.AddChartObject(currentLabel6);

ChartText currentLabel7 = new ChartText(pTransform1, theLabelFont, "I(b) = 350uA",
    15.5, y1[6,85]+1,zdepth , ChartObj.PHYS_POS);
chartVu.AddChartObject(currentLabel7);

ChartText regionLabel = new ChartText(pTransform1, theLabelFont,
    "Linear" + "\n" + "Region", 4.0, 40 , zdepth, ChartObj.PHYS_POS);
chartVu.AddChartObject(regionLabel);
```

[VB]

```
' Create a transparent rectangle shape object and position it in 3D space
Dim zdepth As Double = Dataset1.ImplicitZValue
.
.
.
' Place an annotation at each line plot
Dim theLabelFont As New Font("Microsoft Sans Serif", 10, FontStyle.Bold)
Dim currentLabel1 As New ChartText(pTransform1, theLabelFont, "I(b) = 50uA", _
    15.5, y1(0, 85) + 1, zdepth, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel1)

Dim currentLabel2 As New ChartText(pTransform1, theLabelFont, "I(b) = 100uA", _
    15.5, y1(1, 85) + 1, zdepth, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel2)

Dim currentLabel3 As New ChartText(pTransform1, theLabelFont, "I(b) = 150uA", _
    15.5, y1(2, 85) + 1, zdepth, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel3)

Dim currentLabel4 As New ChartText(pTransform1, theLabelFont, "I(b) = 200uA", _
    15.5, y1(3, 85) + 1, zdepth, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel4)

Dim currentLabel5 As New ChartText(pTransform1, theLabelFont, "I(b) = 250uA", _
    15.5, y1(4, 85) + 1, zdepth, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel5)

Dim currentLabel6 As New ChartText(pTransform1, theLabelFont, "I(b) = 300uA", _
    15.5, y1(5, 85) + 1, zdepth, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel6)
```

```

Dim currentLabel7 As New ChartText(pTransform1, theLabelFont, "I(b) = 350uA", _
    15.5, yl(6, 85) + 1, zdepth, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel7)

Dim regionLabel As New ChartText(pTransform1, _
    theLabelFont, "Linear" + ControlChars.Lf + "Region", 4.0, 40, _
    zdepth, ChartObj.PHYS_POS)
chartVu.AddChartObject(regionLabel)

```

ChartText time coordinates example (extracted from the example program MiscCharts.LineGap)

[C#]

```

Font theLabelFont = new Font("SansSerif", 14, FontStyle.Bold);
ChartText chartLabel1 = new ChartText(pTransform1,
    theLabelFont, "Sales", xValues[1].GetCalendarMsecs(),
    groupBarData[1,1], ChartObj.PHYS_POS);
chartLabel1.SetColor(Color.White);
chartLabel1.SetYJust(ChartObj.AXIS_MIN);
chartVu.AddChartObject(chartLabel1);

```

[Visual Basic]

```

Dim theLabelFont As New Font("SansSerif", 14, FontStyle.Bold)
Dim chartLabel1 As New ChartText(pTransform1, theLabelFont, "Sales", _
    xValues(1).GetCalendarMsecs(), groupBarData(1, 1), ChartObj.PHYS_POS)
chartLabel1.SetColor(Color.White)
chartLabel1.SetYJust(ChartObj.AXIS_MIN)
chartVu.AddChartObject(chartLabel1)

```

Chart Title Classes

Class ChartTitle

ChartText

|

+--ChartTitle

The **ChartTitle** class creates a header, subheader or footer for a chart. The most common constructors are:

ChartTitle constructors

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal tfont As Font, _
    ByVal tstring As String _
)

Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal tfont As Font, _
    ByVal tstring As String, _
    ByVal nttitletype As Integer, _
    ByVal nttitlepos As Integer _
)

Overloads Public Sub New( _
    ByVal tfont As Font, _
    ByVal tstring As String _
)

Overloads Public Sub New( _
    ByVal tfont As Font, _
    ByVal tstring As String, _
    ByVal nttitletype As Integer, _
    ByVal nttitlepos As Integer _
)

[C#]
public ChartTitle(
    PhysicalCoordinates transform,
    Font tfont,
    string tstring
);

public ChartTitle(
    PhysicalCoordinates transform,
    Font tfont,
    string tstring,
    int nttitletype,
    int nttitlepos
);

public ChartTitle(
    Font tfont,
    string tstring
);

public ChartTitle(
    Font tfont,
    string tstring,
    int nttitletype,
    int nttitlepos
);

```

transform

Places the text in the coordinate system defined by transform.

<i>tfont</i>	A reference to a Font object.
<i>tstring</i>	A reference to a string object.
<i>ntitletype</i>	The title can be a header, subhead or footer. Use one of the title type constants: CHART_HEADER, CHART_SUBHEAD or CHART_FOOTER.
<i>ntitlepos</i>	The title can be centered with respect to the entire graph area volume, or the plot area volume. Use one of the title position constants: CENTER_GRAPH or CENTER_PLOT.

See the special note, *Special note about ChartText rotations* at the beginning of this chapter.

ChartTitle example (extracted from the example program SimpleLinePlots.LineFill)

[C#]

```

.
.
.
// Chart title
Font theTitleFont = new Font("Microsoft Sans Serif", 16, FontStyle.Bold);
mainTitle = new ChartTitle( theTitleFont, "Profits are Expected to Rise");
mainTitle.SetTitleType(ChartObj.CHART_HEADER);
mainTitle.SetTitlePosition( ChartObj.CENTER_GRAPH);
mainTitle.SetColor(Color.White);
chartVu.AddChartObject(mainTitle);

// Chart footer
Font theFooterFont = new Font("Microsoft Sans Serif", 9, FontStyle.Bold);
footer = new ChartTitle( theFooterFont, "Ribbon plots can be filled, or unfilled.
Each plot segment can be assigned a unique color.");
footer.SetTitleType(ChartObj.CHART_FOOTER);
footer.SetTitlePosition( ChartObj.CENTER_GRAPH);
footer.SetColor(Color.Black);
chartVu.AddChartObject(footer);

```

[Visual Basic]

```

.
.
.
Dim theTitleFont As New Font("SansSerif", 16, FontStyle.Bold)

```

```

mainTitle = New ChartTitle(theTitleFont, _
    "Profits are Expected to Rise")
mainTitle.SetTitleType(ChartObj.CHART_HEADER)
mainTitle.SetTitlePosition(ChartObj.CENTER_GRAPH)
mainTitle.SetColor(Color.White)
chartVu.AddChartObject(mainTitle)

Dim theFooterFont As New Font("SansSerif", 10, FontStyle.Bold)
footer = New ChartTitle(theFooterFont, _
    "Graphs can have background gradients, semi-transparent colors, legends,
    titles and data tooltips.")
footer.SetTitleType(ChartObj.CHART_FOOTER)
footer.SetTitlePosition(ChartObj.CENTER_GRAPH)
footer.SetTitleOffset(8)
footer.SetColor(Color.White)
chartVu.AddChartObject(footer)

```

Class AxisTitle

ChartText

```

|
+--AxisTitle

```

The **AxisTitle** class creates a title for an axis. The text is horizontal for x-axis titles and vertical for y-axis titles. The most common constructor is:

AxisTitle Constructor

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal axis As Axis, _
    ByVal thefont As Font, _
    ByVal s As String _
)

[C#]
public AxisTitle(
    Axis axis,
    Font thefont,
    string s
);

```

axis The base axis this title is associated with.

thefont The font object used to display the axis title.

s Sets the title string.

See the special note, *Special note about ChartText rotations* at the beginning of this chapter.

ChartTitle example (extracted from the example program LabeledDatapoints.LabeledDatapoints)

[C#]

```

.
.
.
LinearAxis yAxis = new LinearAxis(pTransform1, ChartObj.Y_AXIS);
chartVu.AddChartObject(yAxis);

NumericAxisLabels xAxisLab = new NumericAxisLabels(xAxis);
xAxisLab.SetTextFont(theFont);
chartVu.AddChartObject(xAxisLab);

NumericAxisLabels yAxisLab = new NumericAxisLabels(yAxis);
yAxisLab.SetTextFont(theFont);
chartVu.AddChartObject(yAxisLab);

Font titleFont = new Font("SansSerif", 12, FontStyle.Bold);
AxisTitle yaxistitle = new AxisTitle( yAxis, titleFont, "Test Score");
chartVu.AddChartObject(yaxistitle);

AxisTitle xaxistitle = new AxisTitle( xAxis, titleFont, "Student #");
chartVu.AddChartObject(xaxistitle);

```

[Visual Basic]

```

Dim xAxis As New LinearAxis(pTransform1, ChartObj.X_AXIS)
chartVu.AddChartObject(xAxis)

Dim yAxis As New LinearAxis(pTransform1, ChartObj.Y_AXIS)
chartVu.AddChartObject(yAxis)

Dim xAxisLab As New NumericAxisLabels(xAxis)
xAxisLab.SetTextFont(theFont)
chartVu.AddChartObject(xAxisLab)

Dim yAxisLab As New NumericAxisLabels(yAxis)

```

```

yAxisLab.SetTextFont(theFont)
chartVu.AddChartObject(yAxisLab)

Dim titleFont As New Font("SansSerif", 12, FontStyle.Bold)
Dim yaxistitle As New AxisTitle(yAxis, titleFont, "Test Score")
chartVu.AddChartObject(yaxistitle)

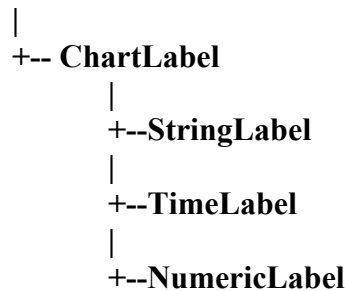
Dim xaxistitle As New AxisTitle(xAxis, titleFont, "Student #")
chartVu.AddChartObject(xaxistitle)

```

Numeric, Time and String Label Classes

Class ChartLabel

ChartText



The **ChartLabel** class is the abstract base class for all of the formatted label classes. The axis label classes use formatted labels to label the axis tick marks. They are also useful for chart annotations. Position the objects using any of the coordinate systems. Rotate and justify the text vertically and horizontally.

ChartLabel constructors

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal tfont As Font, _
    ByVal initialvalue1 As Double, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal npostype As Integer, _
    ByVal nnumformat As Integer, _
    ByVal ndecimal As Integer _
)

Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal tfont As Font, _
    ByVal initialvalue1 As Double, _
    ByVal x As Double, _
    ByVal y As Double, _

```

```

        ByVal npostype As Integer, _
        ByVal nnumformat As Integer, _
        ByVal ndecimal As Integer, _
        ByVal xjust As Integer, _
        ByVal yjust As Integer, _
        ByVal rotation As Double _
    )

    Overloads Public Sub New( _
        ByVal transform As PhysicalCoordinates, _
        ByVal tfont As Font, _
        ByVal initialvalue1 As Double, _
        ByVal x As Double, _
        ByVal y As Double, _
        ByVal z As Double, _
        ByVal npostype As Integer, _
        ByVal nnumformat As Integer, _
        ByVal ndecimal As Integer, _
        ByVal xjust As Integer, _
        ByVal yjust As Integer, _
        ByVal rotation As Double _
    )

    Overloads Public Sub New( _
        ByVal transform As PhysicalCoordinates, _
        ByVal tfont As Font, _
        ByVal initialvalue1 As Double, _
        ByVal x As Double, _
        ByVal y As Double, _
        ByVal z As Double, _
        ByVal npostype As Integer, _
        ByVal nnumformat As Integer, _
        ByVal ndecimal As Integer _
    )

    [C#]
    public NumericLabel(
        PhysicalCoordinates transform,
        Font tfont,
        double initialvalue1,
        double x,
        double y,
        int npostype,
        int nnumformat,
        int ndecimal
    );

    public NumericLabel(
        PhysicalCoordinates transform,
        Font tfont,
        double initialvalue1,
        double x,
        double y,
        int npostype,
        int nnumformat,
        int ndecimal,
        int xjust,
        int yjust,
        double rotation
    );

    public NumericLabel(
        PhysicalCoordinates transform,
        Font tfont,
        double initialvalue1,
        double x,
        double y,

```

```

    double z,
    int npostype,
    int nnumformat,
    int ndecimal,
    int xjust,
    int yjust,
    double rotation
);
public NumericLabel(
    PhysicalCoordinates transform,
    Font tfont,
    double initialvalue1,
    double x,
    double y,
    double z,
    int npostype,
    int nnumformat,
    int ndecimal
);

```

```

NumericLabel(PhysicalCoordinates transform,
              Font tfont, double initialvalue,
              double x, double y, int npostype,
              int nnumformat, int ndecimal)

```

<i>transform</i>	Places the text in the coordinate system defined by transform.
<i>tfont</i>	A reference to a Font object.
<i>initialvalue</i>	The initial value of the numeric label.
<i>x</i>	Specifies the x-value of the text position
<i>y</i>	Specifies the y-value of the text position
<i>z</i>	Specifies the z-value of the text position
<i>npostype</i>	Specifies the if the position of the text is specified in physical coordinates, normalized coordinates or .Net device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.
<i>nnumformat</i>	Specifies the numeric format of the label. Use one of the numeric format constants : DECIMALFORMAT, SCIENTIFICFORMAT, BUSINESSFORMAT, ENGINEERINGFORMAT, PERCENTFORMAT, CURRENCYFORMAT and EXPONENTFORMAT.
<i>ndecimal</i>	The number of digits to display to the right of the decimal point.

<i>xjust</i>	Specifies the horizontal justification of the text. Use one of the text justification constants: JUSTIFY_MIN, JUSTIFY_CENTER or JUSTIFY_MAX.
<i>yjust</i>	Specifies the vertical justification of the text. Use one of the text justification constants: JUSTIFY_MIN, JUSTIFY_CENTER or JUSTIFY_MAX.
<i>rotation</i>	The rotation (-360 to 360 degrees) of the text in the normal viewing plane.

The **TimeLabel** and **StringLabel** classes are similar, unique properties for each are listed below.

See the special note, *Special note about ChartText rotations* at the beginning of this chapter.

TimeLabel constructors

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal date As ChartCalendar, _
    ByVal timeformat As Integer _
)

Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal tfont As Font, _
    ByVal date As ChartCalendar, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal npostype As Integer, _
    ByVal timeformat As Integer, _
    ByVal xjust As Integer, _
    ByVal yjust As Integer, _
    ByVal rotation As Double _
)

Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal tfont As Font, _
    ByVal date As ChartCalendar, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal z As Double, _
    ByVal npostype As Integer, _
    ByVal timeformat As Integer, _
    ByVal xjust As Integer, _
    ByVal yjust As Integer, _
    ByVal rotation As Double _
)
```

```
[C#]
public TimeLabel(
    PhysicalCoordinates transform,
    ChartCalendar date,
    int timeformat
);
```

```
public TimeLabel(
    PhysicalCoordinates transform,
    Font tfont,
    ChartCalendar date,
    double x,
    double y,
    int npostype,
    int timeformat,
    int xjust,
    int yjust,
    double rotation
);
```

```
public TimeLabel(
    PhysicalCoordinates transform,
    Font tfont,
    ChartCalendar date,
    double x,
    double y,
    double z,
    int npostype,
    int timeformat,
    int xjust,
    int yjust,
    double rotation
);
```

date The calendar value used to initialize the label.

timeformat The format used to convert the calendar value to a text string. Use one of the calendar format constants, TIMEDATEFORMAT_XXX.

StringLabel constructors

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal tfont As Font, _
    ByVal tstring As String, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal npostype As Integer _
)

Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal tfont As Font, _
    ByVal tstring As String, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal npostype As Integer, _
    ByVal xjust As Integer, _
```



```

        ByVal yjust As Integer, _
        ByVal rotation As Double _
    )

Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal tfont As Font, _
    ByVal tstring As String, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal z As Double, _
    ByVal npostype As Integer, _
    ByVal xjust As Integer, _
    ByVal yjust As Integer, _
    ByVal rotation As Double _
)

[C#]
public StringLabel(
    PhysicalCoordinates transform,
    Font tfont,
    string tstring,
    double x,
    double y,
    int npostype
);

public StringLabel(
    PhysicalCoordinates transform,
    Font tfont,
    string tstring,
    double x,
    double y,
    int npostype,
    int xjust,
    int yjust,
    double rotation
);

public StringLabel(
    PhysicalCoordinates transform,
    Font tfont,
    string tstring,
    double x,
    double y,
    double z,
    int npostype,
    int xjust,
    int yjust,
    double rotation
);

```

tstring A reference to a string object.

Place text in is a time coordinate system (**TimeCoordinates**) by converting the time x-position to milliseconds and using the milliseconds as the x-position value.

See the special note, *Special note about ChartText rotations* at the beginning of this chapter.

NumericLabel and StringLabel examples (extracted from the example program MouseListeners.MoveDatapoints)

[C#]

```
class1Average = Dataset1.GetAverageY();
class2Average = Dataset2.GetAverageY();

StringLabel class1Label = new StringLabel(pTransform1, subheadFont, "Class #1" +
"\n" + "Average",
    0.9, 0.3, ChartObj.NORM_GRAPH_POS);
chartVu.AddChartObject(class1Label);
class1AverageLabel = new NumericLabel(pTransform1, subheadFont,
class1Average, 0.9, 0.35, ChartObj.NORM_GRAPH_POS,
ChartObj.DECIMALFORMAT,1);
chartVu.AddChartObject(class1AverageLabel);

StringLabel class2Label = new StringLabel(pTransform1, subheadFont,
    "Class #2" + "\n" + "Average",
    0.9, 0.5, ChartObj.NORM_GRAPH_POS);
chartVu.AddChartObject(class2Label);
class2AverageLabel = new NumericLabel(pTransform1, subheadFont,
class2Average, 0.9, 0.55, ChartObj.NORM_GRAPH_POS, ChartObj.DECIMALFORMAT,1);
chartVu.AddChartObject(class2AverageLabel);
```

[Visual Basic]

```
class1Average = Dataset1.GetAverageY()
class2Average = Dataset2.GetAverageY()

Dim class1Label As New StringLabel(pTransform1, _
    subheadFont, "Class #1" + ControlChars.Lf + "Average", _
    0.9, 0.3, ChartObj.NORM_GRAPH_POS)
chartVu.AddChartObject(class1Label)
class1AverageLabel = New NumericLabel(pTransform1, subheadFont, _
    class1Average, 0.9, 0.35, ChartObj.NORM_GRAPH_POS, _
    ChartObj.DECIMALFORMAT, 1)
```

```
chartVu.AddChartObject(class1AverageLabel)

Dim class2Label As New StringLabel(pTransform1, subheadFont, _
    "Class #2" + ControlChars.Lf + "Average", _
    0.9, 0.5, ChartObj.NORM_GRAPH_POS)
chartVu.AddChartObject(class2Label)
class2AverageLabel = New NumericLabel(pTransform1, _
    subheadFont, class2Average, 0.9, 0.55, _
    ChartObj.NORM_GRAPH_POS, ChartObj.DECIMALFORMAT, 1)
chartVu.AddChartObject(class2AverageLabel)
```


21. Adding Lines, Shapes, Images and Arrows to a Chart

ChartShape

Arrow

ChartImage

It is not possible to take into account every possible graphical object that a programmer wants to add to a graph. Specialized applications require specialized objects. Rather than create a large group of classes that duplicate the functions of the **Arc2D**, **Rectangle3D** and other classes, a generalized class has been created, **ChartShape**, which can place and display in a chart any object that can be expressed as a **System.Drawing.Drawing2D.GraphicsPath**.

The **Arrow** defines an arrow shape useable with the **ChartShape** class. The **Arrow** class creates the arrows in the **ArrowPlot** class, and it can also place individual arrows in a chart. The class creates a base arrow with a custom arrowhead and shaft size. Scale, rotate and position the arrow in a chart.

The **ChartImage** class places a **System.Drawing.Image** object anywhere in a chart. It can be a small element of the chart, inside or outside of the plot area volume, or it can be sized to fill the plot area volume or graph area volume and used as a background object.

Important note about rotations

You will find that sometimes you will want to position a shape or image that is dependent on the chart coordinate system, including the 3D rotational transforms that the coordinate system goes through before it is rendered to the screen. Examples are shapes and images that must maintain a fixed relationship to physical coordinates of the charts plotting area. In this example you use a constructor that has as a parameter the coordinate system that you want the shape or image to rotate with. Other times you will want the opposite to be true; even though the chart can be rotated, you want certain shapes and images to remain fixed with respect to the screen viewport of the **ChartView** window. Examples of this are the shapes used to highlight chart titles and images used as background objects. All of the chart shape and image classes have versions that do NOT require that you pass in a coordinate system as the first parameter. In this case, a default, Cartesian coordinate system is created for that object, and that coordinate system is NOT rotated when the other coordinate systems of the chart are rotated. You can position the chart shape or image using either a plot area normalized (NORM_PLOT_POS) , or graph area normalized (NORM_GRAPH_POS) coordinate system.

Generic Shape Class

Class ChartShape

GraphObj



The **ChartShape** class places arbitrary **GraphicsPath** objects in a chart. If the shape includes absolute positioning information, use (0,0) as the xy position parameters of the shape. If the shape coordinates are relative coordinates with the object centered on (0,0), place the shape at the position you want using the xy position parameters. The xy position parameters are the rotation origin of shape.

ChartShape constructor

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal ashape As GraphicsPath, _
    ByVal shapecoordstype As Integer, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal npositiontype As Integer, _
    ByVal rotation As Integer _
)

Overloads Public Sub New( _
    ByVal ashape As GraphicsPath, _
    ByVal shapecoordstype As Integer, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal npositiontype As Integer, _
    ByVal rotation As Integer _
)

[C#]
public ChartShape(
    PhysicalCoordinates transform,
    GraphicsPath ashape,
    int shapecoordstype,
    double x,
    double y,
    int npositiontype,
    int rotation
);

public ChartShape(
    GraphicsPath ashape,
    int shapecoordstype,
    double x,
    double y,
    int npositiontype,
    int rotation
);

```

<i>transform</i>	The shape object is placed in the coordinate system defined by transform.
<i>ashape</i>	A reference to a System.Drawing.Drawing2D.GraphicsPath object.
<i>shapecoordstype</i>	Specifies if the coordinate system defining the shape is specified in physical coordinates, normalized coordinates or .Net device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.
<i>x</i>	Specifies the x-value of the shape position.
<i>y</i>	Specifies the y-value of the shape position.
<i>npostype</i>	Specifies the if the position of the shape is specified in physical coordinates, normalized coordinates or .Net device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.
<i>rotation</i>	The rotation, in degrees, of the shape in the normal viewing plane. The rotation will take place about the objects (0.0, 0.0) coordinate. If the object is not defined with a center of (0.0, 0.0) it may be rotated out of the current viewing plane.

ChartShape example (extracted from the example program MultiLinePlots.MultiLines)

[C#]

```

ChartView chartVu = new ChartView();
.
.
.
// Create a transparent rectangle shape object and position it in 3D space
double zdepth = Dataset1.ImplicitZValue;;
Color alphaColor = Color.FromArgb(127,170, 100, 50);
ChartAttribute attrib2 = new ChartAttribute (alphaColor, 1,DashStyle.Solid,
alphaColor);
attrib2.SetFillFlag(true);
Rectangle3D linearRegionRect = new Rectangle3D(0.1,0.1, 1.5,50);

```

404 Lines, Shapes, Images and Arrows

```
GraphicsPath rectpath = new GraphicsPath();
rectpath.AddRectangle(linearRegionRect.GetRectangleF());
ChartShape linearRegionShape = new ChartShape(pTransform1, rectpath,
    ChartObj.PHYS_POS, 0.0, 0.0, zdepth, ChartObj.PHYS_POS, 0);
    linearRegionShape.SetChartObjAttributes(attrib2);
chartVu.AddChartObject(linearRegionShape);
```

[Visual Basic]

```
' Create a transparent rectangle shape object and position it in 3D space
Dim zdepth As Double = Dataset1.ImplicitZValue
Dim alphaColor As Color = Color.FromArgb(127, 170, 100, 50)
Dim attrib2 As New ChartAttribute(alphaColor, 1, DashStyle.Solid, alphaColor)
attrib2.SetFillFlag(True)
Dim linearRegionRect As New Rectangle3D(0.1, 0.1, 1.5, 50)
Dim rectpath As New GraphicsPath()
rectpath.AddRectangle(linearRegionRect.GetRectangleF())
Dim linearRegionShape As New ChartShape(pTransform1, rectpath, ChartObj.PHYS_POS,
0.0, 0.0, zdepth, ChartObj.PHYS_POS, 0)
linearRegionShape.SetChartObjAttributes(attrib2)
chartVu.AddChartObject(linearRegionShape)
```

ChartShape example (extracted from the example program LabeledDatapoints.LabeledDatapoints)

[C#]

```
// Define a line that is drawn under the title
GraphicsPath titleLine = new GraphicsPath();
titleLine.AddLine(0.1f, 0.1f, 0.9f, 0.1f);
ChartShape titleLineShape = new ChartShape( titleLine,
    ChartObj.NORM_GRAPH_POS, 0.0, 0.0, chartObj.NORM_GRAPH_POS, 0);
titleLineShape.SetLineWidth(3);
chartVu.AddChartObject(titleLineShape);
```

[Visual Basic]

```
Dim titleLine As New GraphicsPath()
titleLine.AddLine(0.1F, 0.1F, 0.9F, 0.1F)
Dim titleLineShape As New ChartShape( titleLine, _
    ChartObj.NORM_GRAPH_POS, 0.0, 0.0, ChartObj.NORM_GRAPH_POS, 0)
titleLineShape.SetLineWidth(3)
chartVu.AddChartObject(titleLineShape)
```


Chart Image Class

Class ChartImage

GraphObj

|
+-- ChartImage

The **ChartImage** class will place a **System.Drawing.Image** object anywhere in a chart. It can be a small element of the chart, inside or outside of the plot area volume or it can be sized to fill the plot area volume or graph area volume and used as a background object. While the position of the image object can be specified using 3D coordinates, and the image can move with the chart when rotated, the image itself does not undergo 3D transformations and will always be displayed as a flat, and rectangular image, with respect to the screen.

ChartImage constructor

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal aimage As Image, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal z As Double, _
    ByVal npostype As Integer, _
    ByVal rotation As Integer _
)

Overloads Public Sub New( _
    ByVal aimage As Image, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal z As Double, _
    ByVal npostype As Integer, _
    ByVal rotation As Integer _
)

Overloads Public Sub New( _
    ByVal transform As PhysicalCoordinates, _
    ByVal aimage As Image, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal npostype As Integer, _
    ByVal rotation As Integer _
)

Overloads Public Sub New( _
    ByVal aimage As Image, _
    ByVal x As Double, _
    ByVal y As Double, _
    ByVal npostype As Integer, _
    ByVal rotation As Integer _
)
```

```

[C#]
public ChartImage(
    PhysicalCoordinates transform,
    Image aimage,
    double x,
    double y,
    double z,
    int npostype,
    int rotation
);

public ChartImage(
    Image aimage,
    double x,
    double y,
    double z,
    int npostype,
    int rotation
);

public ChartImage(
    PhysicalCoordinates transform,
    Image aimage,
    double x,
    double y,
    int npostype,
    int rotation
);

public ChartImage(
    Image aimage,
    double x,
    double y,
    int npostype,
    int rotation
);

```

<i>transform</i>	The coordinate system for the new ChartImage object.
<i>aimage</i>	A reference to the Image object that is to be placed in the chart.
<i>x</i>	The x-value for the position of the image in the chart.
<i>y</i>	The y-value for the position of the image in the chart.
<i>z</i>	The z-value for the position of the image in the chart.
<i>npostype</i>	Specifies whether the x- and y-position values are specified in normalized coordinates, or physical coordinates. Use one of the position constants: NORM_POS, PHYS_POS.
<i>rotation</i>	The rotation of the image specified in degrees.

ChartImage example (extracted from the example program ImageCharts.ImageBackground)

[C#]

```

// All of this is to try and load image file
String path = Environment.CurrentDirectory;
String filename = "..\\..\\Images\\ChartClouds.jpg";
Bitmap aImage = null;
try
{
    aImage = new Bitmap(filename);
}
catch (System.ArgumentException )
{
    filename = "Images\\ChartClouds.jpg";
    aImage = null;
}
if (aImage == null)
{
    try
    {
        aImage = new Bitmap(filename);
    }
    catch (System.ArgumentException )
    {
        aImage = null;
    }
}
// If image file loaded, create a ChartImage object and add it to the chart sized
for the full graph area
if (aImage != null)
{
    // don't reference the transform so it does not get rotated
    ChartImage chartImage = new ChartImage(
        aImage, 0, 0, ChartObj.NORM_GRAPH_POS, 0 );
    chartImage.SetSizeMode(ChartObj.COORD_SIZE);
    chartImage.SetImageSize(new Dimension3D(1,1));
    chartVu.AddChartObject(chartImage);
}

```

[Visual Basic]

```

Dim filename As [String] = "..\\Images\\ChartClouds.jpg"
Dim aImage As Bitmap
Try

```

```

        aImage = New Bitmap(filename)
    Catch err As System.ArgumentException
        filename = "Images\ChartClouds.jpg"
        aImage = Nothing
    End Try
If (aImage Is Nothing) Then
    Try
        aImage = New Bitmap(filename)
        Catch err As System.ArgumentException
            aImage = Nothing
        End Try
    End If

    If Not (aImage Is Nothing) Then
        Dim chartImage As New ChartImage(pTransform1, aImage, 0, 0, _
            ChartObj.NORM_GRAPH_POS, 0)
        chartImage.SetSizeMode(ChartObj.COORD_SIZE)
        chartImage.SetImageSize(New Dimension3D(1, 1))
        chartVu.AddChartObject(chartImage)
    End If

```

Generic Arrow Class

Class Arrow

ChartObj

```

|
+-- Arrow

```

The **Arrow** defines an arrow shape useable with the **ChartShape** class. The **Arrow** class creates the arrows in the **ArrowPlot** class, and it can also place individual arrows in a chart. The class creates a base arrow with a custom arrowhead and shaft size. Scale, rotate and position the arrow in a chart. The arrow is defined using device coordinates.

Arrow constructor

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal arrowshafthalfwidth As Double, _
    ByVal arrayshaftlength As Double, _
    ByVal arrowheadhalfwidth As Double, _
    ByVal arrowheadlength As Double _
)

[C#]
public Arrow(
    double arrowshafthalfwidth,
    double arrayshaftlength,

```

```

    double arrowheadhalfwidth,
    double arrowheadlength
);

```

<i>arrowshafthalfwidth</i>	Sets the half-width of the arrow shaft. (default 1)
<i>arrayshaftlength</i>	Sets the length of the arrow shaft. (default 7)
<i>arrowheadhalfwidth</i>	Sets the half-width of the arrow head. (default 2)
<i>arrowheadlength</i>	Sets the length of the arrow head. (default 3)

The default arrow has a length of about 10 pixels and a width of 4 pixels at the head. The size of the various parts can be set to whatever values you want to create an arrow of with an aspect ratio appropriate to your application. You can scale the arrow by setting the **ArrowScaleFactor** property. Get a **GraphicsPath** object defining the arrow shape by calling the **GetArrowShape** method.

Arrow example (extracted from the example program **MultiLinePlots.MultiLines**)

[C#]

```

// Create an arrow shaped object and position it in 3D space
Arrow regionArrow = new Arrow(1,40,6,15);
ChartAttribute arrowAttrib = new ChartAttribute (Color.Black, 1,DashStyle.Solid,
Color.Black);
arrowAttrib.SetFillFlag(true);
ChartShape arrowShape = new ChartShape(pTransform1, regionArrow.GetArrowShape(),
ChartObj.DEV_POS, 1.5, 40.0,zdepth, ChartObj.PHYS_POS,195);
arrowShape.SetChartObjAttributes(arrowAttrib);
chartVu.AddChartObject(arrowShape);

```

[Visual Basic]

```

' Create an arrow shaped object and position it in 3D space
Dim regionArrow As New Arrow(1, 40, 6, 15)
Dim arrowAttrib As New ChartAttribute(Color.Black, 1, DashStyle.Solid,
Color.Black)
arrowAttrib.SetFillFlag(True)

```

410 *Lines, Shapes, Images and Arrows*

```
Dim arrowShape As New ChartShape(pTransform1, _  
    regionArrow.GetArrowShape(), ChartObj.DEV_POS, 1.5, _  
    40.0, zdepth, ChartObj.PHYS_POS, 195)  
arrowShape.SetChartObjAttributes(arrowAttrib)  
chartVu.AddChartObject(arrowShape)
```


22. File and Printer Rendering Classes

ChartPrint BufferedImage

High quality B&W and color printing is an important feature of the charting library. The resulting graph renders on the printer using the resolution of the output device, for both text and graphical elements of the chart, and does not transfer a grainy image from the computer to the printer. The **QCChart3D** for .Net software uses the Microsoft .Net **PrintDocument** component to implement printing.

The **BufferedImage** class converts a chart into a .Net **Bitmap** object, or saves the chart to a file in any of the graphics formats supported by the **System.Drawing.Imaging.ImageFormat** class. The image file is placeable in a web page or an application program.

You can print one or more **ChartView** derived objects to a single page, positioning each **ChartView** object on the printed page.

Printing a Chart

Class ChartPrint

ChartObj

|
+--ChartPrint

The **ChartPrint** class uses the Microsoft .Net **PrintDocument** component to implement printing. The class selects, setups, and outputs one or more charts to a printer.

ChartPrint constructor for printing one ChartView graph per page

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal component As ChartView, _
    ByVal nsizeMode As Integer _
)

[C#]
public ChartPrint(
    ChartView component,
    int nsizeMode
);
```

<i>component</i>	Specifies the ChartView object to be printed.
<i>nsizemode</i>	Specifies the printer mapping mode. Use one of the mapping mode constants:
PRT_MAX	Print the view so that paper is used maximally. Text prints proportional to other objects, aspect ratio is maintained.
PRT_EXACT	Print the view at the same size as the screen, at least as far as .Net maintains a one to one coorespondence in the printing engine. The aspect ratio of the view is maintained.
PRT_RECT	Print the view to the specified rectangle, specified using the SetPrintRect method and normalized coordinates. Regardless of the print rectangle, the aspect ratio of the chart is maintained.

ChartPrint constructor for printing multiple ChartView graphs per page

```
[Visual Basic]
Overloads Public Sub New( _
    ByVal components As ChartView(), _
    ByVal posrects As Rectangle3D() _
)

[C#]
public ChartPrint(
    ChartView[] components,
    Rectangle3D[] posrects
);
```

<i>components</i>	Specifies an array of ChartView objects to be printed.
<i>posrects</i>	An array of position rectangles (Rectangle3D objects) that place the related ChartView graph on the printed page. Only the x, y, width, and height properties of the position rectangle are used. The position rectangle values are expressed in normalized (0.0 to 1.0) printer, output page coordinates. The value 0.0 is the upper left corner, and the value 1.0 is the lower right corner of the printed page. Graphs must still retain their aspect ratio, so no matter what the size of the positioning rectangle, either the height or the width of the positioning rectangle will end up being the limiting factor controlling the page rendered size of the graph.

Call the **ChartPrint.DoPrintDialog** method after creating the **ChartPrint** object. Then call the **ChartPrint.DoPrintPage** method, rendering the chart to the printer. If the **DoPrintDialog** method is not called prior to **DoPrintPage**, the **DoPrintPage** method automatically invokes the **DoPrintDialog** method. Subsequent calls to **DoPrintPage** will not invoke the **DoPrintDialog** method.

Using All of the Paper When Printing

The PRT_MAX mode prints the chart as large as possible, while maintaining the same aspect ratio as the original **ChartView**. If the width is the limiting factor, the bottom of the printed page will always be blank. The same is true of the PRT_RECT mode. While the PRT_RECT mode can control the size and position of the chart on the printed page, it cannot change the aspect ratio of the chart.

The only way to fill the printed page in portrait or landscape mode is establish the screen **ChartView** size with the same aspect ratio as the 8 1/2 x 11 printed page printable area (about 6.5 x 9 assuming the 1 inch default margins). Assuming portrait mode, a **ChartView** sized to 650W x 900H will fill the page, as will other **ChartView** sizes with the same proportions (500W x 692H, 400W x 554H, 300W x 415 etc.). If you are printing in landscape mode then the chart width and height values would be swapped.

ChartPrint example, rendering a single graph to a printed page (extracted from the example program LinePlotSalesVolume.LinePlotSalesVolume)

```
// This routine invokes the chart objects PageSetupItem method
public void PageSetup(object sender, System.EventArgs e)
{
    ChartView chartVu = this;
    if (chartVu != null)
    {
        if (printobj == null)
        {
            printobj = new ChartPrint(chartVu);
        }
        else
            printobj.PrintChartView = chartVu;
        printobj.PageSetupItem(sender, e);
    }
}

// This routine invokes the chart objects printer setup dialog method
```

```

public void PrinterSetup(object sender, System.EventArgs e)
{
    ChartView chartVu = this;
    if (chartVu != null)
    {
        if (printobj == null)
        {
            printobj = new ChartPrint(chartVu);
        }
        else
            printobj.PrintChartView = chartVu;
        printobj.DoPrintDialog();
    }
}

// This routine invokes the chart objects PrintPreviewItem method
public void PrintPreview(object sender, System.EventArgs e)
{
    ChartView chartVu = this;
    if (chartVu != null)
    {
        if (printobj == null)
        {
            printobj = new ChartPrint(chartVu);
        }
        else
            printobj.PrintChartView = chartVu;
        printobj.PrintPreviewItem(sender, e);
    }
}

// This routine prints a chart by invoking the chart objects DocPrintPage method
public void PrintPage(object sender, System.EventArgs e)
{
    ChartView chartVu = this;
    if (chartVu != null)
    {
        if (printobj == null)
        {
            printobj = new ChartPrint(chartVu);
            printobj.DoPrintDialog();
        }
        else
    
```

```

        printobj.PrintChartView = chartVu;

        printobj.DocPrintPage( sender, e);
    }
}

```

[Visual Basic]

```

' This routine invokes the chart objects PageSetupItem method
Public Sub PageSetup(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim chartVu As ChartView = Me
    If Not (chartVu Is Nothing) Then
        If printobj Is Nothing Then
            printobj = New ChartPrint(chartVu)
        Else
            printobj.PrintChartView = chartVu
        End If
        printobj.PageSetupItem(sender, e)
    End If
End Sub 'PageSetup

' This routine invokes the chart objects printer setup dialog method
Public Sub PrinterSetup(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim chartVu As ChartView = Me
    If Not (chartVu Is Nothing) Then
        If printobj Is Nothing Then
            printobj = New ChartPrint(chartVu)
        Else
            printobj.PrintChartView = chartVu
        End If
        printobj.DoPrintDialog()
    End If
End Sub 'PrinterSetup

' This routine invokes the chart objects PrintPreviewItem method
Public Sub PrintPreview(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim chartVu As ChartView = Me
    If Not (chartVu Is Nothing) Then
        If printobj Is Nothing Then
            printobj = New ChartPrint(chartVu)

```

```

Else
    printobj.PrintChartView = chartVu
End If
printobj.PrintPreviewItem(sender, e)
End If
End Sub 'PrintPreview

' This routine prints a chart by invoking the chart objects DocPrintPage method
Public Sub PrintPage(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim chartVu As ChartView = Me
    If Not (chartVu Is Nothing) Then
        If printobj Is Nothing Then
            printobj = New ChartPrint(chartVu)
            printobj.DoPrintDialog()
        Else
            printobj.PrintChartView = chartVu
        End If
        printobj.DocPrintPage(sender, e)
    End If
End Sub 'PrintPage

```

ChartPrint example, four ChartView objects to a single printed page (extracted from the example program WebStatistics.Form1)

[C#]

```

ChartPrint printobj = null;
.
.
.
private void InitializePrintChart()
{
    double width = 0.5;
    double height = 0.5;
    double depth = 0.0;

    ChartView [] components = {viewsAndVisits1, bandwidthUsage1,
        searchEngines1, searchPhrases1};
    Rectangle3D [] placement =
        {new Rectangle3D(0.0, 0.0, 0.0, width, height, depth ),
        new Rectangle3D(0.5, 0.0, 0.0, width, height, depth ),

```

```

        new Rectangle3D(0.0, 0.50, 0.0, width, height, depth ),
        new Rectangle3D(0.5, 0.50, 0.0, width, height, depth ));
    if (printobj == null)
    {
        printobj = new ChartPrint(components, placement);
    }
}

private void menuItem2_Click(object sender, System.EventArgs e)
{
    if (printobj == null)
        InitializePrintChart();
    printobj.PageSetupItem(sender, e);
}

private void menuItem3_Click(object sender, System.EventArgs e)
{
    if (printobj == null)
        InitializePrintChart();
    printobj.DoPrintDialog();
}

private void menuItem4_Click(object sender, System.EventArgs e)
{
    if (printobj == null)
        InitializePrintChart();
    printobj.PrintPreviewItem(sender, e);
}

private void menuItem5_Click(object sender, System.EventArgs e)
{
    if (printobj == null)
        InitializePrintChart();
    printobj.DocPrintPage( sender, e);
}

```

[VB]

```

Dim printobj As ChartPrint
.
.
.

```

420 *File and Printer Rendering*

```
Private Sub InitializePrintChart()
    Dim inset As Double = 0.025
    Dim width As Double = 0.45
    Dim height As Double = 0.45
    Dim depth As Double = 0.0

    Dim components As ChartView() = {viewsAndVisits1, bandwidthUsage1, _
        searchEngines1, searchPhrases1}
    Dim placement As Rectangle3D() = {New Rectangle3D(inset, inset, 0.0, width, _
        height, depth), New Rectangle3D(0.5 + inset, inset, 0.0, width, height, _
        depth), New Rectangle3D(inset, 0.5 + inset, 0.0, width, height, depth), _
        New Rectangle3D(0.5 + inset, 0.5 + inset, 0.0, width, height, depth)}
    If printobj Is Nothing Then
        printobj = New ChartPrint(components, placement)
    End If
End Sub 'InitializePrintChart

Private Sub MenuItem2_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MenuItem2.Click
    If (printobj Is Nothing) Then
        InitializePrintChart()
    End If
    printobj.DoPrintDialog()
End Sub

Private Sub MenuItem3_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MenuItem3.Click
    If (printobj Is Nothing) Then
        InitializePrintChart()
    End If
    printobj.PageSetupItem(sender, e)
End Sub

Private Sub MenuItem4_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MenuItem4.Click
    If (printobj Is Nothing) Then
        InitializePrintChart()
    End If
    printobj.PrintPreviewItem(sender, e)
End Sub

Private Sub MenuItem5_Click(ByVal sender As System.Object, _
```



```

ByVal e As System.EventArgs) Handles MenuItem5.Click
    If (printobj Is Nothing) Then
        InitializePrintChart()
    End If
    printobj.DocPrintPage(sender, e)
End Sub

```

Capturing the Chart as a Buffered Image

Class BufferedImage

ChartObj

```

|
+-- BufferedImage

```

The **BufferedImage** class creates a **Bitmap** object that is used to render a **ChartView** object into an image buffer. The rendering takes place when the **BufferedImage.Render** method or **BufferedImage.SaveImage** method is called.

BufferedImage constructor

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal component As ChartView, _
    ByVal imgformat As ImageFormat _
)

```

```

[Visual Basic]
Overloads Public Sub New( _
    ByVal component As ChartView _
)

```

```

[C#]
public BufferedImage(
    ChartView component,
    ImageFormat imgformat
);

```

```

public BufferedImage(
    ChartView component
);

```

component The **ChartView** object that is the source for the chart image.

imageformat An image format object specifying the format of the rendered image. Use one of the .Net ImageFormat constants: ImageFormat.Bmp, ImageFormat.Jpeg, ImageFormat.Tif,

ImageFormat.Gif, ImageFormat.Png, ImageFormat.Emf, or
ImageFormat.Wmf

The **BufferedImage.GetBufferedImage** method converts the chart to the .Net **Bitmap** object specified by the imageformat object and returns a reference the resulting bitmap.

BufferedImage example (extracted from the example program LinePlotSalesVolume.LinePlotSalesVolume)

[C#]

```
public void SaveAsFile(object sender, System.EventArgs e)
{
    ChartView chartview = this;
    String filename = this.Name;
    SaveFileDialog imagefilechooser = new SaveFileDialog();
    imagefilechooser.Filter =
        "Image
Files (*.BMP;*.JPG;*.GIF;*.TIFF;*.PNG) | *.BMP;*.JPG;*.GIF;*.TIFF;*.PNG | All files
(*.*) | *.*";
    imagefilechooser.FileName = filename;
    if (imagefilechooser.ShowDialog() == DialogResult.OK)
    {
        filename = imagefilechooser.FileName;
        FileInfo fileinformation = new FileInfo(filename);
        String fileext = fileinformation.Extension;
        fileext = fileext.ToUpper();
        ImageFormat fileimageformat;
        if (fileext == ".BMP" )
            fileimageformat = ImageFormat.Bmp;
        else if ((fileext == ".JPG") || (fileext == ".JPEG"))
            fileimageformat = ImageFormat.Jpeg;
        else if ((fileext == ".GIF"))
            fileimageformat = ImageFormat.Gif;
        else if ((fileext == ".TIF") || (fileext == ".TIFF"))
            fileimageformat = ImageFormat.Tiff;
        else if ((fileext == ".PNG"))
            fileimageformat = ImageFormat.Png;
        else
            fileimageformat = ImageFormat.Bmp;

        BufferedImage savegraph = new BufferedImage(chartview, fileimageformat);
```

```

        savegraph.Render();
        savegraph.SaveImage(filename);
    }
}

```

[Visual Basic]

```

Public Sub SaveAsFile(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim chartview As ChartView = Me
    Dim filename As [String] = Me.Name
    Dim imagefilechooser As New SaveFileDialog()
    imagefilechooser.Filter = "Image
Files (*.BMP;*.JPG;*.GIF;*.TIFF;*.PNG)|*.BMP;*.JPG;*.GIF;*.TIFF;*.PNG|All files
(*.*)|*.*"
    imagefilechooser.FileName = filename
    If imagefilechooser.ShowDialog() = DialogResult.OK Then
        filename = imagefilechooser.FileName
        Dim fileinformation As New FileInfo(filename)
        Dim fileext As [String] = fileinformation.Extension
        fileext = fileext.ToUpper()
        Dim fileimageformat As ImageFormat
        If fileext = ".BMP" Then
            fileimageformat = ImageFormat.Bmp
        Else
            If fileext = ".JPG" Or fileext = ".JPEG" Then
                fileimageformat = ImageFormat.Jpeg
            Else
                If fileext = ".GIF" Then
                    fileimageformat = ImageFormat.Gif
                Else
                    If fileext = ".TIF" Or fileext = ".TIFF" Then
                        fileimageformat = ImageFormat.Tiff
                    Else
                        If fileext = ".PNG" Then
                            fileimageformat = ImageFormat.Png
                        Else
                            fileimageformat = ImageFormat.Bmp
                        End If
                    End If
                End If
            End If
        End If
        Dim savegraph As New BufferedImage(chartview, fileimageformat)

```

424 *File and Printer Rendering*

```
        savegraph.Render()  
        savegraph.SaveImage(filename)  
    End If  
End Sub 'SaveAsFile
```

23. Using QCChart3D for .Net to Create Windows Applications

(*** Critical Note ***) Running the Example Programs

The example programs for **QCChart3D** charting software are supplied in complete source. In order to save space, they have not been pre-compiled which means that many of the intermediate object files needed to view the main form are not present. This means that **ChartView** derived control will not be visible on the main Form if you attempt to view the main form before the project has been compiled. The default state for all of the example projects should be the Start Page. Before you do view any other file or form, do a build of the project. This will cause the intermediate files to be built. If you attempt to view the main Form before building the project, Visual Studio decides that the **ChartView** control placed on the main form does not exist and deletes it from the project.

The primary view class of the **QCChart3D** library is the **ChartView** class. The **ChartView** class is derived from the .Net **System.Windows.Forms.UserControl** class. It has the properties and methods of the underlying **UserControl** class.

Follow the following steps in order to incorporate the **QCChart3D** classes into your program. This is not the only way to add charts to an application. In general, any technique that works with **UserControl** derived classes will work. We found the technique described below this to be the most flexible.

Visual Basic for .Net

If you do not already have an application program project, create one using the Visual Studio project wizard (**File | New | Project | Visual Basic Projects | Windows Application**). On the left select a project type of **Visual Basic Projects**. Give the project a unique name (our version of this example is **UserChartExample1**). You will end with a basic Form based application. For purposes of this example, the chart will placed in the initial, default form.

- Add a User Control class to the project (**Project | Add User Control**). Enter a class name of **UserChartControl1** (We choose **UserChartControl2** to keep it from conflicting with our own **UserChartControl1** example).
- Right click on Reference in the Solution Explorer window and select **Add Reference**. Browse to the Quinn-Curtis/DotNet/lib subdirectory and select the **QCChart3DNet.DLL**.

View the **UserChartControl2.vb** code. Change the **Inherits** line to **com.quinncurtis.chart3dnet.ChartView**. This adds a local version of the control to the project. The form code should now look like:

```
Public Class UserChartControl2
    Inherits com.quinncurtis.chart3dnet.ChartView

End Class
```

- **Critical Step:** Make sure you add the following lines to the top of the **UserChartControl2.vb** code to resolve the **QCChart3D** and other graphics classes used in the example.

```
Imports com.quinncurtis.chart3dnet
Imports System.Drawing
Imports System.Drawing.Drawing2D
```

- The step of creating and defining a User Control that inherits from **com.quinncurtis.chart3dnet.ChartView** only needs to be performed once. Any instance of the control that you add to any form in the project will derive from **UserChartControl2**.
- Build the Solution (**Build | Build Solution**). This will compile the **UserChartControl2** class and make it accessible as a component on the Toolbox and to derive from. If the project fails to compile you need to go back and check the previous steps.
- You can create as many custom chart controls as your application requires. Each custom chart control will inherit from the **com.quinncurtis.chart3dnet.ChartView** control. Or
- (Optional) You can create inherited controls from the **UserChartControl2** class that you already created. Create an inherited control by selecting **Project | Add Inherited Control**. Give the inherited control a unique name, i.e. **UserChartInheritedControl1**. When you select Open, choose **UserChartControl2** in the Inheritance Picker. The result is new control added to the project. Build the solution and the **UserChartInheritedControl1** control will be added to the Toolbox in addition to the **UserChartControl2**.
- Look at the **UserChartControl2** class. The chart is created in the **InitializeChart** method. Until this method is called, the **UserChartControl2** appears as an empty shell. You can call this method from somewhere outside of the class to avoid problems associated debugging errors in user controls at design time.

- Go to the main form, **Form1**. Go to the toolbox and select the **UserChartControl2** from the Windows Forms list. Drop it onto the main form and size it.
- Double click on the **UserChartControl2** in the main **Form1** form. This will add a **UserChartControl21_Load** method to the **Form1** class. The chart initialization code will be invoked by this event

```
Private Sub UserChartControl21_Load(ByVal sender _
    As System.Object, ByVal e As System.EventArgs) Handles _
    UserChartControl21.Load
    UserChartControl21.InitializeChart()
End Sub
```

- Define the chart by customizing the **UserChartControl2.InitializeChart** method. Our example has copied the initialization code from the **SimpleScatter** class in the **ScatterPlots** example.
- You should be able to compile the project without error. No chart will be visible yet. The **Form1_Load** will only be called at runtime.
- You should now be able to compile, run and view the entire project. Any changes you make in the **UserChartControl2** form will be reflected in the application.

Visual C# for .Net

If you do not already have an application program project, create one using the Visual Studio project wizard (**File | New | Project | Visual C# Projects | Windows Application**). On the left select a project type of **Visual C# Projects**. Give the project a unique name (our version of this example is **UserChartExample1**). You will end with a basic Form based application. For purposes of this example, the chart will placed in the initial, default form.

- Add a User Control class to the project (**Project | Add User Control**). Enter a class name of **UserChartControl2** (We choose **UserChartControl2** to keep it from conflicting with our own **UserChartControl1** example).
- Right click on Reference in the Solution Explorer window and select **Add Reference**. Browse to the Quinn-Curtis/DotNet/lib subdirectory and select the **QCChart3DNet.DLL**.

- View the **UserChartControl2.cs** code. Change the base class of **UserChartControl2** to **com.quinncurtis.chart3dnet.ChartView**. This adds a local version of the control to the project. The C# form code should now look like:

```
namespace UserChartExample1
{
    /// <summary>
    /// Summary description for UserChartView.
    /// </summary>
    public class UserChartControl2: com.quinncurtis.chart3dnet.ChartView
    {
        .
        .
        .
    }
    .
    .
    .
}
```

- **Critical Step:** Make sure you add the following lines to the top of the **UserChartControl2.cs** code to resolve the QCChart3D and other graphics classes used in the example.

```
using System.Drawing.Drawing;
using System.Drawing.Drawing2D;
using com.quinncurtis.chart3dnet;
```

- The step of creating and defining a User Control that inherits from **com.quinncurtis.chart3dnet.ChartView** only needs to be performed once. Any instance of the control that you add to any form in the project will derive from **UserChartControl2**.
- Build the Solution (**Build | Build Solution**). This will compile the **UserChartControl2** class and make it accessible as a component on the Toolbox and to derive from. If the project fails to compile you need to go back and check the previous steps.
- You can create as many custom chart controls as your application requires. Each custom chart control will inherit from the **com.quinncurtis.chart3dnet.ChartView** control. Or

- (Optional) You can create inherited controls from the **UserChartControl2** class that you already created. Create an inherited control by selecting **Project | Add Inherited Control**. Give the inherited control a unique name, i.e. **UserChartInheritedControl1**. When you select Open, choose **UserChartControl2** in the Inheritance Picker. The result is new control added to the project. Build the solution and the **UserChartInheritedControl1** control will be added to the Toolbox in addition to the **UserChartControl2**.
- Right click on the **UserChartControl2** form and view the underlying C# code. We placed all of the chart customization code in the **InitializeChart** method. Until this method is called, the **UserChartControl2** appears as an empty shell. Call this method from somewhere outside of the class to avoid problems associated debugging errors in user controls at design time.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
using com.quinncurtis.chart3dnet;

namespace UserChartExample1
{
    public class UserChartControl2 : com.quinncurtis.chart3dnet.ChartView
    {
        private System.ComponentModel.IContainer components = null;

        public UserChartControl2()
        {
            // This call is required by the Windows Form Designer.
            InitializeComponent();
            // TODO: Add any initialization after the InitializeComponent call
        }

        public void InitializeChart()
        {
            ChartView chartVu = this;

        }
    }
}
```

```

    }
}

```

- Go to the main form, **Form1**. Go to the toolbox and select the **UserChartControl2** from the Windows Forms list. Drop it onto the main form and size it.
- Double click on the **UserChartControl2** in the main **Form1** form. This will add a **Form1_Load** method to the **Form1** class. The chart initialization code will be invoked by this event

```

private void UserChartControl21_Load(object sender,
                                     System.EventArgs e)
{
    UserChartControl21.InitializeChart();
}

```

- Define the chart by customizing the **UserChartControl2.InitializeChart** method. Our example has copied the initialization code from the **SimpleScatter** class in the **ScatterPlots** example.
- You should be able to compile the project without error. No chart will be visible yet. The **Form1_Load** will only be called at runtime.
- You should now be able to compile, run and view the entire project. Any changes you make in the **UserChartControl2** form will be reflected in the application.

24. Using QCChart3D for .Net to Create Web Applications

There are two ways you can integrate the **QCChart3D** library in your web pages: the first is to serve up an image file created using **QCChart3D** from the server; and the second is to actually embed a **QCChart3D** UserControl, derived from our **ChartView** class, in the HTML web page.

In the first method, you use a **Image** component in either a HTML web page, or in an ASP.Net page. The source URL for the image component can either be an image file, or a URL address to an ASP page that serves up the image file as a stream. The **QCChart3D** library includes a **BufferedImage** class that renders a **ChartView** chart as a memory bitmap and will output that image to a file or a stream. The image file or stream can be in any image format supported by the .Net **System.Drawing.Imaging.ImageFormat** class.

The second method, embedding a **QCChart3D ChartView** derived object in an HTML page, is made possible because we eliminated the license file from the software. There was no way to check the license file from a control embedded in an HTML page without generating exceptions that would invalidate the control. We aren't sure what browsers support this, other than Microsoft Internet Explorer. In order to guarantee the necessary security clearance to run in the browser you must install our QCChart3DNet.DLL library in the GAC of the client computer – more on that later.

Included with the software is the complete source to the three web applications we have running on our web site at:

<http://quinn-curtis.com/Chart3DAspWebApp.htm>

<http://quinn-curtis.com/Chart3DWebApplication1.htm>

<http://quinn-curtis.com/Chart3DHTMLAspApp.htm>

The source is found in our DotNet/QCChart3D/Visual CSharp/Examples/Asp.Net and our DotNet/QCChart3D/Visual Basic/Examples/Asp.Net folders.

Do Not Try and Compile our ASP.Net Applications

Unlike the Windows applications created using Visual Studio .Net, we have found it extremely difficult to move ASP.Net applications to a different Visual Studio .Net environment. Visual Studio splits the files that it creates between the computer hosting

Visual Studio and the server. The projects include all sorts of references to the server where the application is expected to run. Obviously our server is not your server. If you are an expert at moving ASP.Net projects give it a try. Your best bet is to develop your own shell application for your own server, get it to run, and then add Web Forms as you need them to support the chart application. Once you create a shell ASP.Net application that you know runs on the target server, adding a chart using the behind code of a ASP.Net Web Form is simple.

On Demand Creation of QCChart3D Images from a Server

Visual C# for .Net

If you do not already have an application program project, create one using the Visual Studio project wizard (**File | New | Project | Visual C# Projects | ASP.Net Application**). On the left select a project type of Visual C# Projects. Give the project a unique name (our version of this example is **Chart3DWebApplication1**). You will end with a basic WebForm based application with one **System.Web.UI.Page** derived form (WebForm1.aspx by default). For purposes of this example, the chart will placed in the initial, default form. Assume that you want to make an ASP.Net application that streams an image file to an image component in a web page.

You do not need to add anything to the WebForm1.aspx page. Right click on the page and select View Code. This brings up the code behind file (WebForm1.aspx.cs) for the WebForm1.aspx page. Add the following references in the using section at the top of the WebForm1.aspx.cs file:

```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
using com.quinncurtis.chart3dnet;
```

- Right click on Reference in the Solution Explorer window and select **Add Reference**. Browse to the Quinn-Curtis/DotNet/lib subdirectory and select the **QCChart3DNet.DLL**. Also, add **System.Windows.Forms.dll** from the .Net list box.

Add a method that makes the chart and returns a **ChartView** object of the chart. In the example below this is the function **GetInitializeChart**. It has String arguments that contain the x- and y-data values, and integers that specify the desired height and width for the chart.

```

using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using com.quinncurtis.chart3dnet;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;

namespace Chart3DWebApplication1
{
    /// <summary>
    /// Summary description for WebForm1.
    /// </summary>
    public class WebForm1 : System.Web.UI.Page
    {
        private void Page_Load(object sender, System.EventArgs e)
        {
            // Get Input from HTML ASP page request
            String widthstring = Request.Params.Get(0);
            String heightstring = Request.Params.Get(1);
            String XDataString = Request.Params.Get(2);
            String YDataString = Request.Params.Get(3);

            int imagewidth = int.Parse(widthstring);
            int imageheight = int.Parse(heightstring);

            ChartView chartVu = GetInitializeChart(XDataString, YDataString,
                                                    imagewidth , imageheight);

            // Stream chart back as a JPeg image
            BufferedImage chartimage = new BufferedImage(chartVu,
                                                         ImageFormat.Jpeg);
            // get your binary image data here
            Response.ContentType = "image/jpeg";
            chartimage.SaveImage(Response.OutputStream);
        }
    }
}

```

```

private ChartView GetInitializeChart(String XDataString,
    String YDataString, int imagewidth, int imageheight)
{
    double [] xvalues = {0,1,2,3,4,5};
    double [] yvalues = {3,7,2,5,11,13};

    ChartView chartVu = new ChartView();

    String [] xdata = XDataString.Split(',');
    String [] ydata = YDataString.Split(',');
    for (int i=0; i < xdata.Length; i++)
    {
        xvalues[i] = double.Parse(xdata[i]);
        yvalues[i] = double.Parse(ydata[i]);
    }
    // Set size of view to match the size of target image for best rendering
    chartVu.Size = new Size(imagewidth, imageheight);
    Font theFont = new Font("SansSerif", 10, FontStyle.Bold);

    SimpleDataset Dataset1 = new SimpleDataset("First", xvalues, yvalues);

    CartesianCoordinates pTransform1 = new CartesianCoordinates();
    pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_NEAR,
        ChartObj.AUTOAXES_NEAR);
    pTransform1.SetGraphBorderDiagonal(0.15, .15, .90, 0.90) ;
    Background graphbackground1 = new Background( pTransform1,
        ChartObj.GRAPH_BACKGROUND, Color.White);
    chartVu.AddChartObject(graphbackground1);

    LinearAxis xAxis1 = new LinearAxis(pTransform1, ChartObj.X_AXIS);
    xAxis1.SetColor(Color.Black);
    chartVu.AddChartObject(xAxis1);

    LinearAxis yAxis1 = new LinearAxis(pTransform1, ChartObj.Y_AXIS);
    yAxis1.SetColor(Color.Red);
    chartVu.AddChartObject(yAxis1);

    NumericAxisLabels xAxisLab1 = new NumericAxisLabels(xAxis1);
    xAxisLab1.SetColor(Color.Black);
    chartVu.AddChartObject(xAxisLab1);

    NumericAxisLabels yAxisLab1 = new NumericAxisLabels(yAxis1);

```

```

yAxisLab1.SetColor(Color.Red);
chartVu.AddChartObject(yAxisLab1);

ChartAttribute attrib1 = new ChartAttribute(Color.Red,
    3,DashStyle.Solid);
SimpleLinePlot thePlot1 = new SimpleLinePlot(pTransform1, Dataset1,
    attrib1);
chartVu.AddChartObject(thePlot1);

Font theTitleFont = new Font("SansSerif", 18,  FontStyle.Bold);
String GraphTitle = "XData = " + XDataString;
ChartTitle mainTitle = new ChartTitle(pTransform1, theTitleFont,
    GraphTitle);
mainTitle.AddNewLineTextString("YData = " + YDataString);
mainTitle.SetTitleType(ChartObj.CHART_HEADER);
mainTitle.SetTitlePosition( ChartObj.CENTER_GRAPH);
mainTitle.SetColor(Color.Black);
chartVu.AddChartObject(mainTitle);

return chartVu;
}

#region Web Form Designer generated code
}
}

```

- Modify the Page_Load method that was included by default when the behind code page was created. In the example below parameters that were appended to the ASP page request are retrieved using the Request.Params.Get method. The values define the size and data values of the graph. This is just one way to pass data from the HTML page to the ASP page. In some of our other examples we use a key string that specifies which stocks we want to plot. Using the behind code page you can query some database and get the data values from there, without passing the data through the HTML page. In our Chart3DHTMLAspApp and Chart3DAspWebApp examples, we assume the data is generated on the server. Once the chart is create and available as a **ChartView** object, use the BufferedImage class to render the chart as a JPEG file and stream the image back in the response stream of the requesting HTML page.

```

private void Page_Load(object sender, System.EventArgs e)
{
    // Get Input from HTML ASP page request

```

```

String widthstring = Request.Params.Get(0);
String heightstring = Request.Params.Get(1);
String XDataString = Request.Params.Get(2);
String YDataString = Request.Params.Get(3);

int imagewidth = int.Parse(widthstring);
int imageheight = int.Parse(heightstring);

ChartView chartVu = GetInitializeChart(XDataString, YDataString,
                                       imagewidth , imageheight);

// Stream chart back as a Jpeg image
BufferedImage chartimage = new BufferedImage(chartVu,
                                             ImageFormat.Jpeg);
Response.ContentType = "image/jpeg";
chartimage.SaveImage(Response.OutputStream);
}

```

- Build the Solution (**Build | Build Solution**) and resolve any errors that might have crept in. This example was not designed to be a standalone application. You can try and call the application from Internet Explorer using a URL address that mimics the expected invoking format:

<http://quinn-curtis.com/Chart3DWebApplication1/WebForm1.aspx?WIDTH=666&HEIGHT=448&XDATA=1,2,3,4,5,6&YDATA=3,7,5,11,13,15>

While not particularly useful, if you type in everything correctly it should display the chart defined in the Chart3DWebApplication1 ASP page.

- You will probably want to create a HTML page to host the call to the ASP page. This can be done using FrontPage, a simple editor and the Visual Studio IDE. A simple HTML page that connects the URL address above to an HTML image tag appears below. The complete source is the file HTMLPage1.htm found in the Chart3DWebApplication1 project folder.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <TITLE></TITLE>
    <META content="Microsoft FrontPage 5.0" name="GENERATOR">
  </HEAD>
  <BODY>
    <FORM id="Form1" name="simplegraph">
      <P></P>

```



```

        <IMG height="448" src="http://quinn-
curtis.com/Chart3DWebApplication1/WebForm1.aspx?WIDTH=666&HEIGHT=448&XDATA=1,2,3,4
,5,6&YDATA=3,7,5,11,13,15" width="666" name="thegraph">
        <P></P>
        <p><input type="button" value="Update Image" name="B1"
onclick="updategraph();" ID="Button1">
        &nbsp;
        <P><input type="text" name="T1" size="152" ID="Text1"></P>
        <P>&nbsp;</P>
        <P>HTML Page Containing Graph Image</P>
    </FORM>
</BODY>
</HTML>

```

****** Note: The button onclick event in the HTML example above is attached to updategraph JavaScript function defined below. *******

The HTML page combines an image tag displaying the chart, a button and a text field. Because the URL address of the ASP application is hard coded as the image source URL, the chart is static because the data values do not change. Adding some JavaScript code will make the chart dynamic, creating new data (random in this case) and graphing the results.

```

<SCRIPT LANGUAGE="JavaScript">
var xdata = [1.0,2.0,3.0,4.0,5.0,6.0];
var ydata = [5.0,2.0,7.0,11.0,7.0,13.0];
var xdatastring = "1.0,2.0,3.0,4.0,5.0,6.0";
var ydatastring = "5.0,2.0,7.0,11.0,7.0,13.0";

function updategraph() {
    makenewdata();
    makedatastrings();
    var urlstring = "http://quinn-curtis.com/Chart3DWebApplication1/WebForm1.aspx?"
+
        "WIDTH=" + document.simplegraph.thegraph.width.toString() +
        "&HEIGHT=" + document.simplegraph.thegraph.height.toString() +
        "&XDATA=" + xdatastring +
        "&YDATA=" + ydatastring;
    document.simplegraph.T1.value = urlstring;
    document.simplegraph.thegraph.src= urlstring;
}

function makedatastrings()
{ var i=0;
    xdatastring = "";
    ydatastring = "";

```

```
for (i=0; i < 5; i++)
{
    xdatastring = xdatastring + xdata[i].toFixed(1) + ",";
    ydatastring = ydatastring + ydata[i].toFixed(1) + ",";
} // Add last values without comma
xdatastring = xdatastring + xdata[5].toFixed(1);
ydatastring = ydatastring + ydata[5].toFixed(1);
}

function makenewdata()
{
    var i=0;
    for (i=0; i < 6; i++)
    {
        xdata[i] = i;
        ydata[i] = 15.0 * Math.random();
    }
}

</SCRIPT>
```

Clicking the button will invoke a chart update using new, random data. The new image source URL is displayed in the HTML text box. Display the URL of the HTML page and you should see a chart, a button and a text field.



HTML Page Containing Graph Image

- Another method is to use two WebForms in the ASP application. The first form is setup much like the previous example. The aspx page contains nothing, and the aspx.cs behind code page does all the work, defining a chart and streaming the result back as a JPEG image. The second WebForm page is used to host a collection of Web controls place on the aspx page. One of these Web controls would be an Image web control. The behind code page for this WebForm would connect the URL of the Image control to the first WebForm page, which does nothing be serve up charts. Our example Chart3DASPWebApp uses this approach. The reason you might want to take this approach is that it is much easier to wire up the controls on a WebForm page using the aspx.cs behind code

as opposed to doing it on an HTML page. The drawback is that the entire page is redrawn when anything is posted, causing a momentary flash when the chart is redraw. Using the HTML approach the chart is redrawn without the flash, making it a better method for real-time applications.

Visual Basic for .Net

If you do not already have an application program project, create one using the Visual Studio project wizard (**File | New | Project | Visual Basic Projects | ASP.Net Application**). On the left select a project type of Visual C# Projects. Give the project a unique name (our version of this example is **VBChart3DWebApplication1**). You will end with a basic WebForm based application with one **System.Web.UI.Page** derived form (WebForm1.aspx by default). For purposes of this example, the chart will placed in the initial, default form. Assume that you want to make an ASP.Net application that streams an image file to an image component in a web page.

You do not need to add anything to the WebForm1.aspx page. Right click on the page and select View Code. This brings up the code behind file (WebForm1.aspx.cs) for the WebForm1.aspx page. Add the following references in the using section at the top of the WebForm1.aspx.cs file:

```
Imports com.quinncurtis.chart3dnet
Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports System.Drawing.Imaging
```

- Right click on Reference in the Solution Explorer window and select **Add Reference**. Browse to the Quinn-Curtis/DotNet/lib subdirectory and select the **QCChart3DNet.DLL**. Also add **System.Windows.Forms.dll** from the .Net list box.

Add a method that makes the chart and returns a **ChartView** object of the chart. In the example below this is the function **GetInitializeChart**. It has String arguments that contain the x- and y-data values, and integers that specify the desired height and width for the chart.

```
Public Class WebForm1
    Inherits System.Web.UI.Page

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Get Input from HTML ASP page request
        Dim widthstring As String = Request.Params.Get(0)
        Dim heightstring As String = Request.Params.Get(1)
        Dim XDataString As String = Request.Params.Get(2)
        Dim YDataString As String = Request.Params.Get(3)

        Dim imagewidth As Integer = Integer.Parse(widthstring)
        Dim imageheight As Integer = Integer.Parse(heightstring)

        Dim chartVu As ChartView = GetInitializeChart(XDataString, YDataString,
imagewidth, imageheight)
        ' Stream chart back as a Jpeg image
        Dim ChartImage As BufferedImage = New BufferedImage(chartVu,
ImageFormat.Jpeg)
        ' get your binary image data here
        Response.ContentType = "image/jpeg"
        ChartImage.SaveImage(Response.OutputStream)
    End Sub

    Private Function GetInitializeChart(ByVal XDataString As String, ByVal
YDataString As String, ByVal imagewidth As Integer, ByVal imageheight As Integer)
As ChartView

        Dim xvalues() As Double = {0, 1, 2, 3, 4, 5}
        Dim yvalues() As Double = {3, 7, 2, 5, 11, 13}

        Dim chartVu As ChartView = New ChartView()

        Dim xdata() As String = XDataString.Split(",")
        Dim ydata() As String = YDataString.Split(",")
        Dim i As Integer
        For i = 0 To xdata.Length - 1
            xvalues(i) = Double.Parse(xdata(i))
            yvalues(i) = Double.Parse(ydata(i))
        Next
        ' Set size of view to match the size of target image for best rendering
        chartVu.Size = New Size(imagewidth, imageheight)
        Dim theFont As Font = New Font("SansSerif", 10, FontStyle.Bold)
```

```

    Dim Dataset1 As SimpleDataset = New SimpleDataset("First", xvalues,
yvalues)

    Dim pTransform1 As CartesianCoordinates = New CartesianCoordinates()
    pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_NEAR,
ChartObj.AUTOAXES_NEAR)
    pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 0.9)
    Dim graphbackground1 As Background = New Background(pTransform1,
ChartObj.GRAPH_BACKGROUND, Color.White)
    chartVu.AddChartObject(graphbackground1)

    Dim xAxis1 As LinearAxis = New LinearAxis(pTransform1, ChartObj.X_AXIS)
    xAxis1.SetColor(Color.Black)
    chartVu.AddChartObject(xAxis1)

    Dim yAxis1 As LinearAxis = New LinearAxis(pTransform1, ChartObj.Y_AXIS)
    yAxis1.SetColor(Color.Red)
    chartVu.AddChartObject(yAxis1)

    Dim xAxisLab1 As NumericAxisLabels = New NumericAxisLabels(xAxis1)
    xAxisLab1.SetColor(Color.Black)
    chartVu.AddChartObject(xAxisLab1)

    Dim yAxisLab1 As NumericAxisLabels = New NumericAxisLabels(yAxis1)
    yAxisLab1.SetColor(Color.Red)
    chartVu.AddChartObject(yAxisLab1)

    Dim attrib1 As ChartAttribute = New ChartAttribute(Color.Red, 3,
DashStyle.Solid)
    Dim thePlot1 As SimpleLinePlot = New SimpleLinePlot(pTransform1, Dataset1,
attrib1)
    chartVu.AddChartObject(thePlot1)

    Dim theTitleFont As Font = New Font("SansSerif", 18, FontStyle.Bold)
    Dim GraphTitle As String = "XData = " + XDataString
    Dim mainTitle As ChartTitle = New ChartTitle(pTransform1, theTitleFont,
GraphTitle)
    mainTitle.AddNewLineTextString("YData = " + YDataString)
    mainTitle.SetTitleType(ChartObj.CHART_HEADER)
    mainTitle.SetTitlePosition(ChartObj.CENTER_GRAPH)
    mainTitle.SetColor(Color.Black)
    chartVu.AddChartObject(mainTitle)

    Return chartVu

```

```
End Function
End Class
```

- Modify the Page_Load method that was included by default when the behind code page was created. In the example below parameters that were appended to the ASP page request are retrieved using the Request.Params.Get method. The values define the size and data values of the graph. This is just one way to pass data from the HTML page to the ASP page. In some of our other examples we use a key string that specifies which stocks we want to plot. Using the behind code page you can query some database and get the data values from there, without passing the data through the HTML page. In our Chart3DHTMLAspApp and Chart3DAspWebApp examples, we query the Yahoo database of historical stock values and get the data from them. Once the chart is create and available as a **ChartView** object, use the BufferedImage class to render the chart as a JPEG file and stream the image back in the response stream of the requesting HTML page.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    ' Get Input from HTML ASP page request
    Dim widthstring As String = Request.Params.Get(0)
    Dim heightstring As String = Request.Params.Get(1)
    Dim XDataString As String = Request.Params.Get(2)
    Dim YDataString As String = Request.Params.Get(3)

    Dim imagewidth As Integer = Integer.Parse(widthstring)
    Dim imageheight As Integer = Integer.Parse(heightstring)

    Dim chartVu As ChartView = GetInitializeChart(XDataString, YDataString,
imagewidth, imageheight)
    ' Stream chart back as a Jpeg image
    Dim ChartImage As BufferedImage = New BufferedImage(chartVu,
ImageFormat.Jpeg)
    ' get your binary image data here
    Response.ContentType = "image/jpeg"
    ChartImage.SaveImage(Response.OutputStream)
End Sub
```

- Build the Solution (**Build | Build Solution**) and resolve any errors that might have crept in. This example was not designed to be a standalone application. You can try and call the application from Internet Explorer using a URL address that mimics the expected invoking format:

<http://quinn-curtis.com/VBChart3DWebApplication1/WebForm1.aspx?WIDTH=666&HEIGHT=448&XDATA=1,2,3,4,5,6&YDATA=3,7,5,11,13,15>

While not particularly useful, if you type in everything correctly it should display the chart defined in the VBChart3DWebApplication1 ASP page.

- You will probably want to create a HTML page to host the call to the ASP page. This can be done using FrontPage, a simple editor and the Visual Studio IDE. A simple HTML page that connects the URL address above to an HTML image tag appears below. The complete source is the file HTMLPage1.htm found in the VBChart3DWebApplication1 project folder.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <TITLE></TITLE>
    <META content="Microsoft FrontPage 5.0" name="GENERATOR">
  </HEAD>
  <BODY>
    <FORM id="Form1" name="simplegraph">
      <P></P>
      <IMG height="448" src="http://quinn-curtis.com/VBChart3DWebApplication1/WebForm1.aspx?WIDTH=666&HEIGHT=448&XDATA=1,2,3,4,5,6&YDATA=3,7,5,11,13,15" width="666" name="thegraph">
      <P></P>
      <p><input type="button" value="Update Image" name="B1"
onclick="updategraph();" ID="Button1">
      &nbsp;
      <P><input type="text" name="T1" size="152" ID="Text1"></P>
      <P>&nbsp;</P>
      <P>HTML Page Containing Graph Image</P>
    </FORM>
  </BODY>
</HTML>
```

****** Note: The button onclick event in the HTML example above is attached to updategraph JavaScript function defined below. ******

The HTML page combines an image tag displaying the chart, a button and a text field. Because the URL address of the ASP application is hard coded as the image source URL, the chart is static because the data values do not change. Adding some JavaScript code will make the chart dynamic, creating new data (random in this case) and graphing the results.

```
<SCRIPT LANGUAGE="JavaScript">
var xdata = [1.0,2.0,3.0,4.0,5.0,6.0];
```



```

var ydata = [5.0,2.0,7.0,11.0,7.0,13.0];
var xdatastring = "1.0,2.0,3.0,4.0,5.0,6.0";
var ydatastring = "5.0,2.0,7.0,11.0,7.0,13.0";

function updategraph() {
    makenewdata();
    makedatastrings();

    var urlstring = "http://quinn-
curtis.com/VBChart3DWebApplication1/WebForm1.aspx?" +
        "WIDTH=" + document.simplegraph.thegraph.width.toString() +
        "&HEIGHT=" + document.simplegraph.thegraph.height.toString() +
        "&XDATA=" + xdatastring +
        "&YDATA=" + ydatastring;

    document.simplegraph.T1.value = urlstring;
    document.simplegraph.thegraph.src= urlstring;
}

function makedatastrings()
{ var i=0;
    xdatastring = "";
    ydatastring = "";
    for (i=0; i < 5; i++)
    { xdatastring = xdatastring + xdata[i].toFixed(1) + ",";
      ydatastring = ydatastring + ydata[i].toFixed(1) + ",";
    } // Add last values without comma
    xdatastring = xdatastring + xdata[5].toFixed(1);
    ydatastring = ydatastring + ydata[5].toFixed(1);
}

function makenewdata()
{ var i=0;
    for (i=0; i < 6; i++)
    { xdata[i] = i;
      ydata[i] = 15.0 * Math.random();
    }
}

</SCRIPT>

```

Clicking the button will invoke a chart update using new, random data. The new image source URL is displayed in the HTML text box. Display the URL of the HTML page and you should see a chart, a button and a text field.



HTML Page Containing Graph Image

- Another method is to use two WebForms in the ASP application. The first form is setup much like the previous example. The aspx page contains nothing, and the aspx.cs behind code page does all the work, defining a chart and streaming the result back as a JPEG image. The second WebForm page is used to host a collection of Web controls place on the aspx page. One of these Web controls would be an Image web control. The behind code page for this WebForm would connect the URL of the Image control to the first WebForm page, which does nothing be serve up charts. Our example Chart3DASPWebApp uses this approach. The reason you might want to take this approach is that it is much

easier to wire up the controls on a WebForm page using the aspx.cs behind code as opposed to doing it on an HTML page. The drawback is that the entire page is redrawn when anything is posted, causing a momentary flash when the chart is redraw. Using the HTML approach the chart is redrawn without the flash, making it a better method for real-time applications.

Embedding a QCChart3D Object into Internet Explorer

In order to embed a QCChart3D object into Internet Explorer, the client must have the QCChart3DNet.DLL installed in the system GAC (Global Assembly Cache). This gives the DLL the necessary security clearance to run in a browser. Perform the following steps.

- Locate the Gacutil.exe (found in your Visual Studio installation) program and copy it to our Quinn-Curtis\DotNet\Lib.
- Run Gacutil.exe from the DOS prompt while you are in the Quinn-Curtis\DotNet\Lib directory using the following command line:

```
Gacutil -i QCChart3DNet.DLL
```

You will find an endless source of information about the GAC on the web and there is no need to repeat it. Here is a starting point:

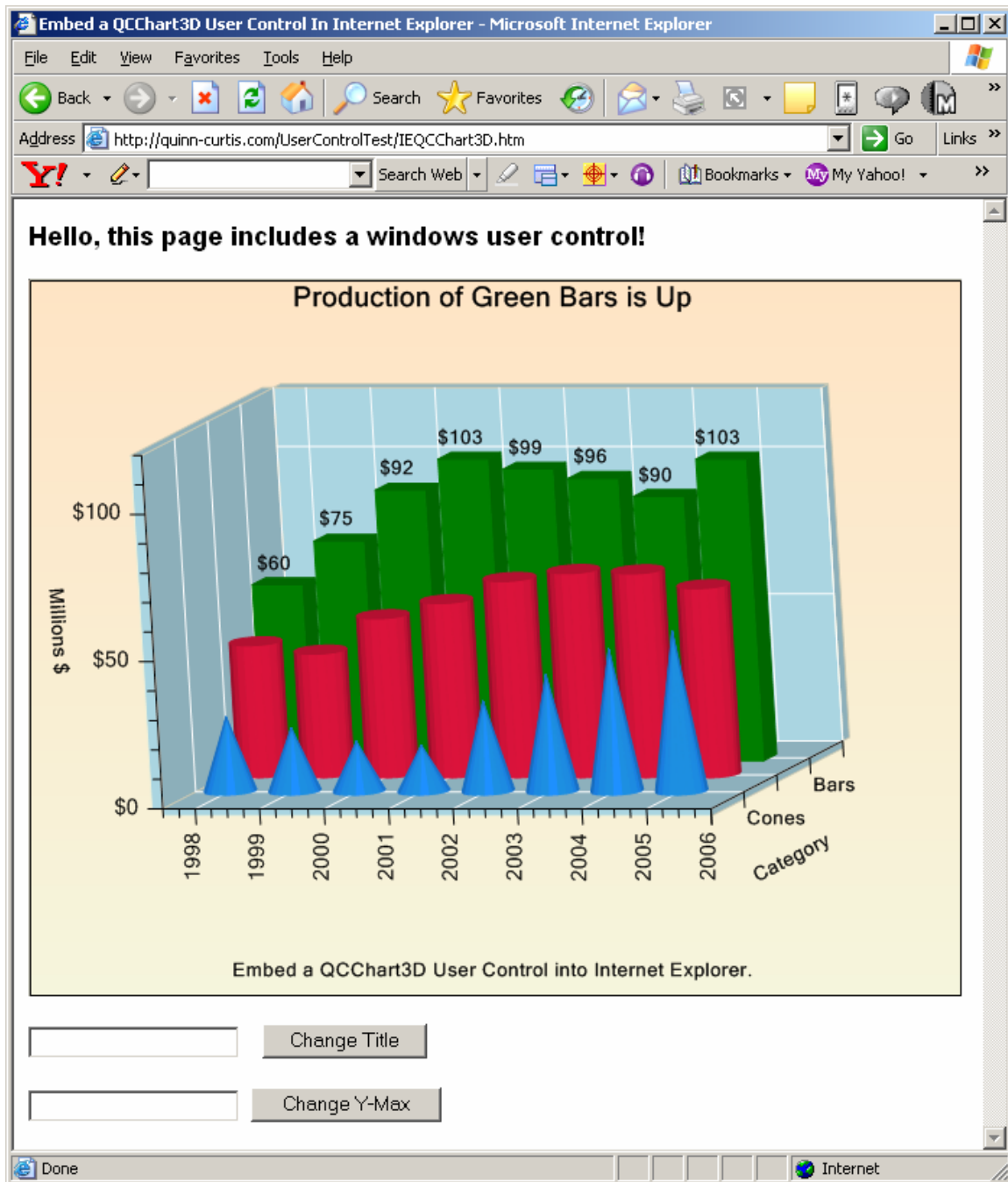
<http://www.codeproject.com/dotnet/DemystifyGAC.asp>

In general, you should not be installing any of our .Net DLL's in the system GAC, unless you have an overriding reason to so. In general, our .Net DLLs should just be distributed as part of an application program and would normally reside in the same folder as the application programs executable. One exception is if you expect a **QCChart3D** derived UserControl to run in Internet Explorer. In that case the QCChart3DNet.DLL must be installed as part of the GAC, so that it has the necessary security clearance, and so that IE can find it when it has to display the control.

If you have installed it correctly, you should be able to view the web page below that references a QCChart3D derived user control.

<http://quinn-curtis.com/UserControlTest/IEQCChart3D.htm>

Once the QCChart3DNet DLL has been successfully installed in the system GAC, you must create a **ChartView** derived **UserControl** that creates the chart you want to display in the browser. This can be done using either C# or VB. And you must create a HTML page that displays the chart, and provides whatever interaction you expect.



Visual C# - Create a Windows Control Library and User Control

You need to create a **ChartView** derived user control that defines your chart, and includes all of the properties that you need for interacting with your chart. This is done almost exactly the same as described in the previous chapter, **Using QCChart3D for .Net to Create Windows Applications**. The difference is that instead of creating a .Net application, resulting in an exe file you can run from Windows, you are instead creating a Windows Control Library, which results in a DLL file. What you end up with is a user control, not an executable program. The user control must be referenced in a specific way

in an HTML file before you can display it. The basic steps for creating the **ChartView** derived control are described below.

Start by creating a Windows Control Library project, using the Visual Studio project wizard (**File | New | Project | Visual C# Projects | Windows Control Library**). On the left select a project type of **Visual C# Projects**. Give the project a unique name (our version of this example is **WindowsControlLibrary1**). You will end with the shell of a UserControl class with the name **UserControl1**. You will modify **UserControl1** and turn it into an object derived from our **ChartView** object.

- Right click on Reference in the Solution Explorer window and select **Add Reference**. Browse to the Quinn-Curtis/DotNet/lib subdirectory and select the **QCChart3DNet.DLL**.
- View the **UserControl1.cs** code. Change the base class of **UserControl1** to **com.quinncurtis.chart3dnet.ChartView**. The C# form code should now look like:

```
namespace WindowsControlLibrary1
{
    /// <summary>
    /// Summary description for UserControl1.
    /// </summary>
    public class UserControl1 : com.quinncurtis.chart3dnet.ChartView
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;

        public UserControl1()
        {
            // This call is required by the Windows.Forms Form Designer.
            InitializeComponent();

            // TODO: Add any initialization after the InitForm call

        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
```

```

    {
        if( disposing )
        {
            if( components != null )
                components.Dispose();
        }
        base.Dispose( disposing );
    }

    .
    .
    .
    }
}

```

- **Critical Step:** Make sure you add the following lines to the top of the **UserChartControl1.cs** code to resolve the QCChart3D and other graphics classes used in the example.

```

using System.Drawing.Drawing2D;
using com.quinncurtis.chart3dnet;

```

- Build the Solution (**Build | Build Solution**). This will compile the **UserControl1** class and generate the **WindowsControlLibrary1.DLL** file. If the project fails to compile you need to go back and check the previous steps.
- You can add as many custom chart controls as you need for the given HTML page you plan to display charts in. Each custom chart control will inherit from the `com.quinncurtis.chart3dnet.ChartView` control. Or
- (Optional) You can create inherited controls from the **UserChartControl1** class that you already created. Create an inherited control by selecting **Project | Add Inherited Control**. Give the inherited control a unique name, i.e. **UserInheritedControl1**. When you select Open, choose **UserControl1** in the Inheritance Picker. The result is new control added to the project. Build the solution and the **UserInheritedControl1** control will be added to the **WindowsControlLibrary1.DLL** in addition to the **UserControl1**.
- View the **UserChartControl1.cs** underlying C# code. We placed all of the chart creation code in the **InitializeChart** method. This method is called in the **UserControl1** constructor, otherwise the **UserControl1** appears as an empty shell. Add whatever properties you need to get data from the HTML file into the chart control (**ChartTitleText** and **YMax** in the example below).

```

using System;

```

```
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;
using com.quinncurtis.chart3dnet;

namespace WindowsControlLibrary1
{
    /// <summary>
    /// Summary description for UserControl1.
    /// </summary>
    public class UserControl1 : ChartView
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;
        ChartTitle mainTitle;
        TimeCoordinates pTransform1;
        LinearAxis yAxis;

        public UserControl1()
        {
            // This call is required by the Windows.Forms Form Designer.
            InitializeComponent();

            // TODO: Add any initialization after the InitForm call
            InitializeChart();
        }

        public void InitializeChart()
        {
            ChartView chartVu = this;

            Font theFont;

            // Simulate data
            int numpnts = 8;
            ChartCalendar []x1= new ChartCalendar[numpnts];
            double [] y1= new double[numpnts];
```

```

        double [] y2= new double[numpnts];
        double [] y3= new double[numpnts];
        .
        .
        .

    }

    ///<summary>
    /// Set/Get the chart title
    ///</summary>
    public string ChartTitleText
    {
        get { return mainTitle.TextString; }
        set { mainTitle.TextString = value;
              this.UpdateDraw();
            }
    }

    ///<summary>
    /// Set/Get the yMax
    ///</summary>
    public double YMax
    {
        get
        {
            return pTransform1.GetScaleStopY(); }
        set
        {
            pTransform1.SetScaleStopY(value);
            yAxis.CalcAutoAxis();
            this.UpdateDraw();
        }
    }
    .
    .
    .
}

```

- Once you can compile the project, copy the resulting WindowsControlLibrary1.DLL file to the desired folder on your web site. In our example we copied the DLL to the <http://quinn-curtis.com/UserControlTest>

folder. The DLL file size should be quite small, since the underlying QCChart3DNet.DLL file is not included as part of the WindowsControlLibrary1.DLL. When Internet Explorer views a web page that properly references the WindowsControlLibrary1.DLL, and its internal **UserControl1** user control, it will download just that DLL. Internet Explorer will not attempt to download the QCChart3DNet.DLL and will look to the system GAC for it instead.

Visual Basic - Create a Windows Control Library and User Control

You need to create a **ChartView** derived user control that defines your chart, and includes all of the properties that you need for interacting with your chart. This is done almost exactly the same as described in the previous chapter, **Using QCChart3D for .Net to Create Windows Applications**. The difference is that instead of creating a .Net application, resulting in an exe file you can run from Windows, you are instead creating a Windows Control Library, which results in a DLL file. What you end up with is a user control, not an executable program. The user control must be referenced in a specific way in an HTML file before you can display it. The basic steps for creating the **ChartView** derived control are described below.

Start by creating a Windows Control Library project, using the Visual Studio project wizard (**File | New | Project | Visual Basic Projects | Windows Control Library**). On the left select a project type of **Visual Basic Projects**. Give the project a unique name (our version of this example is **WindowsControlLibrary1**). You will end with the shell of a UserControl class with the name **UserControl1**. You will modify **UserControl1** and turn it into an object derived from our **ChartView** object.

- Right click on Reference in the Solution Explorer window and select **Add Reference**. Browse to the Quinn-Curtis/DotNet/lib subdirectory and select the **QCChart3DNet.DLL**.
- View the **UserControl1.vb** code. Change the base class of **UserControl1** to **com.quinncurtis.chart3dnet.ChartView**. The VB form code should now look like:

```
Public Class UserControl1
    Inherits com.quinncurtis.chart3dnet.ChartView

    #Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()
    End Sub
End Class
```

```

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'UserControll overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        components = New System.ComponentModel.Container()
    End Sub

#End Region

End Class

```

- **Critical Step:** Make sure you add the following lines to the top of the **UserChartControl1.cs** code to resolve the QCChart3D and other graphics classes used in the example.

```

Imports com.quinncurtis.chart3dnet
Imports System.Drawing
Imports System.Drawing.Drawing2D

```

- Build the Solution (**Build | Build Solution**). This will compile the **UserControl1** class and generate the **WindowsControlLibrary1.DLL** file. If the project fails to compile you need to go back and check the previous steps.

- You can add as many custom chart controls as you need for the given HTML page you plan to display charts in. Each custom chart control will inherit from the `com.quinncurtis.chart3dnet.ChartView` control. Or
- (Optional) You can create inherited controls from the **UserChartControl1** class that you already created. Create an inherited control by selecting **Project | Add Inherited Control**. Give the inherited control a unique name, i.e. **UserInheritedControl1**. When you select Open, choose **UserControl1** in the Inheritance Picker. The result is new control added to the project. Build the solution and the **UserInheritedControl1** control will be added to the `WindowsControlLibrary1.DLL` in addition to the **UserControl1**.
- View the **UserChartControl1.vb** underlying VB code. We placed all of the chart creation code in the **InitializeChart** method. This method is called in the **UserControl1** constructor, otherwise the **UserControl1** appears as an empty shell. Add whatever properties you need to get data from the HTML file into the chart control (**ChartTitleText** and **YMax** in the example below).

```
Imports com.quinncurtis.chart3dnet
Imports System.Drawing
Imports System.Drawing.Drawing2D

Public Class UserControl1
    Inherits com.quinncurtis.chart3dnet.ChartView

    Dim mainTitle As ChartTitle
    Dim pTransform1 As TimeCoordinates
    Dim yAxis As LinearAxis

    #Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call
        InitializeChart()

    End Sub

    Public Sub InitializeChart()
        Dim chartVu As ChartView = Me
```

```

    Dim theFont As Font

    ' Simulate data
    Dim numpnts As Integer = 8
    Dim x1(numpnts - 1) As ChartCalendar
    Dim y1(numpnts - 1) As Double
    Dim y2(numpnts - 1) As Double
    Dim y3(numpnts - 1) As Double
    Dim currentdate As New ChartCalendar(1998, ChartObj.JANUARY, 1)

    .
    .
    .

End Sub 'InitializeChart

'<summary>
'/ Set/Get the chart title
'</summary>

Public Property ChartTitleText() As String
    Get
        Return mainTitle.TextString
    End Get
    Set(ByVal Value As String)
        mainTitle.TextString = value
        Me.UpdateDraw()
    End Set
End Property

'<summary>
'/ Set/Get the yMax
'</summary>

Public Property YMax() As Double
    Get
        Return pTransform1.GetScaleStopY()
    End Get
    Set(ByVal Value As Double)
        pTransform1.SetScaleStopY(value)
        yAxis.CalcAutoAxis()
        Me.UpdateDraw()
    End Set

```

```
End Property
```

```
.  
.
.
```

```
End Class
```

- Once you can compile the project, copy the resulting WindowsControlLibrary1.DLL file to the desired folder on your web site. In our example we copied the DLL to the <http://quinn-curtis.com/UserControlTest> folder. The DLL file size should be quite small, since the underlying QCChart3DNet.DLL file is not included as part of the WindowsControlLibrary1.DLL. When Internet Explorer views a web page that properly references the WindowsControlLibrary1.DLL, and its internal UserControl1 user control, it will download just that DLL. Internet Explorer will not attempt to download the QCChart3DNet.DLL and will look to the system GAC for it instead.

Create the HTML File That Displays the User Control

- Create an HTML file however you normally would do it. You will need to add HTML text to it using some sort of HTML editor. You will find IEQCChart3D.htm, our example, in the WindowsControlLibrary1 folder. The control is referenced in the HTML file as:

```
<OBJECT id="MyTestUserControl1" height="496" width="646"
classid="http://quinn-
curtis.com/UserControlTest/WindowsControlLibrary1.dll#WindowsControlLibrary1.UserC
ontrol1" VIEWASTEXT>
</OBJECT>
```

Notice the *classid* in the <object> tag. The classid has two parts to it. First we have the URL to the DLL file (make sure the DLL is not in the web sites /bin directory). In this example, the WindowsControlLibrary1.DLL file has been explicitly copied to the <http://quinn-curtis.com> web site, and placed in the folder UserControlTest. Hence, the URL to the library is:

<http://quinn-curtis.com/UserControlTest/WindowsControlLibrary1.dll>

It is then followed by a # sign and ends with the fully qualified name of the control (namespace + control name). In this case that is:

```
WindowsControlLibrary1.UserControl1
```


25. Frequently Asked Questions

FAQs

1. Is the **QCChart3D for .Net** software backward compatible with the **Charting Tools for Windows**, the **Graphics Class Libraries for MFC, 3D++, or QCChart2D for .Net**?
2. How do you create a chart with multiple coordinate systems and axes?
3. Can I add new axes, text objects, plot objects, and images to a chart after it is already displayed; or must I create a new chart from scratch taking into account the additional objects?
4. How do you zoom charts that use multiple coordinate systems?
5. How do you select a chart object and create a dialog panel that permits editing of that objects properties?
6. How do you handle missing data points in a chart?
7. How do you update a chart in real-time?
8. How do I prevent flicker when updating my charts on real-time?
9. How do you implement drill down, or data tool tips in a chart?
10. I do not want to my graph to auto-scale. How do I setup the graph axes for a specific range?
11. How do I update my data, and auto-rescale the chart scales and axes to reflect the new data, after it has already been drawn?
12. When I use the auto-scale and auto-axis routines my semi-log chart has the logarithmic axis scaled using powers of 10 (1, 10,100, 1000, etc.) as the starting and ending values, or as the major tick interval for labeling. How do I make my log graphs start at 20 and end at 50,000, with major tick marks at 20, 200, 2000 and 20000?
13. How do I create and use custom, multi-line string labels as the axis labels for my graph?
14. How do I place more than one graph in a view?

15. How do I use your software to generate GIF files?
16. Sometimes the major tick marks of an axis are missing the associated tick mark label ?
17. How do I change the order the chart objects are drawn? For example, I want one of my grid objects to be drawn under the charts line plot objects, and another grid object to be drawn top of the charts line plot objects.
18. How to I use a Forms scrollbar object to control horizontal scrolling of the data in my chart?
19. I am trying to plot 100,000,000 data points and it takes too long to draw the graph. What is wrong with the software and what can I do to make it faster?
20. How do I get data from my database into a chart?
21. How do I use this charting software to generate chart images “on-the-fly”?
22. Can **QCChart3D for .Net** be used to create programs that run like Java applets in web browsers?
23. Are you going to add additional real-time routines to the QCChart3D for .Net library?

1. Is the **QCChart3D for .Net** software backward compatible with the **Charting Tools for Windows**, the **Graphics Class Libraries for MFC, 3D++, or QCChart2D for .Net**?

The **QCChart3D** for .Net software is not backward compatible with earlier Quinn-Curtis non-.Net products. It was developed explicitly for the new .Net programming object oriented programming framework. You should have no problems recreating any charts that you created using our older Windows software; in most cases it will take far fewer lines of code.

The **QCChart3D** API was modeled on the **QCChart2D** API. While not 100% backward compatible with it, it is highly compatible. See the end of Chapter 2 (Class Architecture), the section **Differences between the QCChart2D and QCChart3D API** for discussion about how to convert **QCChart2D** programs to **QCChart3D**.

2. **How do you create a chart with multiple coordinate systems and axes?**

A chart can have as many coordinate systems and axes as you want. A single coordinate system can have one or more x- and/or y-axes. The most common use for multiple axes in a single coordinate system is to place y-axes on both the left and the right sides of a chart, and x-axes above and below. The left and bottom axes usually have numeric or date labels, and the top and right axes just tick marks. This does not have to be the case though; every axis can have axis labels if you want. In general, the axis position in the chart is determined by its intercept. The default value of the intercept is set to the minimums of the coordinate system that the axis is placed in. Adjusting the intercept using the **SetAxisIntercept** method changes the position of the axis in the chart. The axis intercept value is set using units of the coordinate system at right angles to the axis. The example below, extracted from the LineFill example, places y-axes on both the left and right of the chart.

[C#]

```
TimeAxis xAxis = new TimeAxis(pTransform1);
chartVu.AddChartObject(xAxis);

TimeAxis xAxis = new TimeAxis(pTransform1);
xAxis.SetColor(Color.White);
chartVu.AddChartObject(xAxis);

LinearAxis yAxis = new LinearAxis(pTransform1, ChartObj.Y_AXIS);
// Default places y-axis at minimum of x-coordinate scale
yAxis.SetColor(Color.White);
chartVu.AddChartObject(yAxis);

LinearAxis yAxis2 = new LinearAxis(pTransform1, ChartObj.Y_AXIS);
yAxis2.SetAxisIntercept(xAxis.GetAxisMax());
yAxis2.SetAxisTickDir(ChartObj.AXIS_MAX);
yAxis2.SetColor(Color.White);
chartVu.AddChartObject(yAxis2);
```

[VB]

```
Dim xAxis As New TimeAxis(pTransform1)
xAxis.SetColor(Color.White)
chartVu.AddChartObject(xAxis)

Dim yAxis As New LinearAxis(pTransform1, ChartObj.Y_AXIS)
' Default places y-axis at minimum of x-coordinate scale
yAxis.SetColor(Color.White)
chartVu.AddChartObject(yAxis)

Dim yAxis2 As New LinearAxis(pTransform1, ChartObj.Y_AXIS)
```

```

yAxis2.SetAxisIntercept(xAxis.GetAxisMax())
yAxis2.SetAxisTickDir(CharObj.AXIS_MAX)
yAxis2.SetColor(Color.White)
chartVu.AddChartObject(yAxis2)

```

The other common reason to have multiple axes in a chart is to delineate the simultaneous use of different coordinate systems in the chart. In this case each coordinate system has an x- and/or y-axis to differentiate it from the other coordinate systems. When the different coordinate systems are created, they usually overlay the same area of the chart. The default positioning of the axes for each coordinate system will all overlay one another, making the axes unreadable. In the y-axis case you will want to offset additional axes to the left, or to the right of the default axis position, using the **SetAxisIntecept** method. When using the **SetAxisIntercept** method, make sure you specify the position using the units of the coordinate system scale at right angles to the axis. Specify an intercept value outside of the normal scale range to offset the axes so that they do not overlap. The example below, extracted from the `MultipleAxes.MultiAxisChart` example, creates one x-axis, common to all of the charts because the x-scaling for all of the coordinate systems match, and five y-axes, one for each of the five different coordinate systems.

[C#]

```

ChartView chartVu = this;
pTransform1 =
    new CartesianCoordinates( CharObj.LINEAR_SCALE, CharObj.LINEAR_SCALE);
pTransform2 =
    new CartesianCoordinates( CharObj.LINEAR_SCALE, CharObj.LINEAR_SCALE);
pTransform3 =
    new CartesianCoordinates( CharObj.LINEAR_SCALE, CharObj.LINEAR_SCALE);
pTransform4 =
    new CartesianCoordinates( CharObj.LINEAR_SCALE, CharObj.LINEAR_SCALE);
pTransform5 =
    new CartesianCoordinates( CharObj.LINEAR_SCALE, CharObj.LINEAR_SCALE);
.
.
.
SimpleDataset Dataset1 = new SimpleDataset("First", x1,y1,0.5,0.0);
SimpleDataset Dataset2 = new SimpleDataset("Second",x1,y2,0.5,0.0);
SimpleDataset Dataset3 = new SimpleDataset("Third", x1,y3,0.5,0.0);
SimpleDataset Dataset4 = new SimpleDataset("Fourth",x1,y4,0.5,0.0);
SimpleDataset Dataset5 = new SimpleDataset("Fifth", x1,y5,0.5,0.0);

pTransform1.AutoScale(Dataset1, CharObj.AUTOAXES_FAR, CharObj.AUTOAXES_FAR);
pTransform2.AutoScale(Dataset2, CharObj.AUTOAXES_FAR, CharObj.AUTOAXES_FAR);

```

```

pTransform3.AutoScale(Dataset3, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
pTransform4.AutoScale(Dataset4, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
pTransform5.AutoScale(Dataset5, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);

pTransform1.SetGraphBorderDiagonal(0.35, .15, .9, 0.65) ;
pTransform2.SetGraphBorderDiagonal(0.35, .15, .9, 0.65) ;
pTransform3.SetGraphBorderDiagonal(0.35, .15, .9, 0.65) ;
pTransform4.SetGraphBorderDiagonal(0.35, .15, .9, 0.65) ;
pTransform5.SetGraphBorderDiagonal(0.35, .15, .9, 0.65) ;

Background background = new Background( pTransform1, ChartObj.GRAPH_BACKGROUND,
    Color.White);
chartVu.AddChartObject(background);

Background plotbackground = new Background( pTransform1, ChartObj.PLOT_BACKGROUND,
    Color.White);
chartVu.AddChartObject(plotbackground);

ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 2,DashStyle.Solid);
ChartAttribute attrib2 = new ChartAttribute (Color.Red, 2,DashStyle.Solid);
ChartAttribute attrib3 = new ChartAttribute (Color.Green, 2,DashStyle.Solid);
ChartAttribute attrib4 = new ChartAttribute (Color.Orange, 2,DashStyle.Solid);
ChartAttribute attrib5 = new ChartAttribute (Color.Magenta, 2,DashStyle.Solid);

xAxis = new LinearAxis(pTransform1, ChartObj.X_AXIS);
xAxis.SetLineWidth(2);
chartVu.AddChartObject(xAxis);

yAxis1 = new LinearAxis(pTransform1, ChartObj.Y_AXIS);
yAxis1.SetAxisIntercept(0.0);
yAxis1.SetChartObjAttributes(attrib1); // axis color matches line color
chartVu.AddChartObject(yAxis1);

yAxis2 = new LinearAxis(pTransform2, ChartObj.Y_AXIS);
yAxis2.SetAxisIntercept(-18);
yAxis2.SetChartObjAttributes(attrib2); // axis color matches line color
chartVu.AddChartObject(yAxis2);

yAxis3 = new LinearAxis(pTransform3, ChartObj.Y_AXIS);
yAxis3.SetAxisIntercept(-35);
yAxis3.SetChartObjAttributes(attrib3); // axis color matches line color
chartVu.AddChartObject(yAxis3);

```

```

yAxis4 = new LinearAxis(pTransform4, ChartObj.Y_AXIS);
yAxis4.SetAxisIntercept(-52);
yAxis4.SetChartObjAttributes(attrib4); // axis color matches line color
chartVu.AddChartObject(yAxis4);

yAxis5 = new LinearAxis(pTransform5, ChartObj.Y_AXIS);
yAxis5.SetAxisIntercept(xAxis.GetAxisMax());
yAxis5.SetAxisTickDir(ChartObj.AXIS_MAX);
yAxis5.SetChartObjAttributes(attrib5); // axis color matches line color
chartVu.AddChartObject(yAxis5);

NumericAxisLabels xAxisLab = new NumericAxisLabels(xAxis);
xAxisLab.SetTextFont(theFont);
chartVu.AddChartObject(xAxisLab);

NumericAxisLabels yAxisLab1 = new NumericAxisLabels(yAxis1);
yAxisLab1.SetTextFont(theFont);
yAxisLab1.SetAxisLabelsFormat(ChartObj.BUSINESSFORMAT);
chartVu.AddChartObject(yAxisLab1);

NumericAxisLabels yAxisLab2 = new NumericAxisLabels(yAxis2);
yAxisLab2.SetTextFont(theFont);
chartVu.AddChartObject(yAxisLab2);

NumericAxisLabels yAxisLab3 = new NumericAxisLabels(yAxis3);
yAxisLab3.SetTextFont(theFont);
chartVu.AddChartObject(yAxisLab3);

NumericAxisLabels yAxisLab4 = new NumericAxisLabels(yAxis4);
yAxisLab4.SetTextFont(theFont);
chartVu.AddChartObject(yAxisLab4);

NumericAxisLabels yAxisLab5 = new NumericAxisLabels(yAxis5);
yAxisLab5.SetTextFont(theFont);
chartVu.AddChartObject(yAxisLab5);

```

[VB]

```

Dim chartVu As ChartView = Me
pTransform1 = _
    New CartesianCoordinates(ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE)
pTransform2 = _

```

```

        New CartesianCoordinates(ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE)
pTransform3 = _
        New CartesianCoordinates(ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE)
pTransform4 = _
        New CartesianCoordinates(ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE)
pTransform5 = _
        New CartesianCoordinates(ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE)

.
.
.

Dim Dataset1 As New SimpleDataset("First", x1, y1, 0.5, 0.0)
Dim Dataset2 As New SimpleDataset("Second", x1, y2, 0.5, 0.0)
Dim Dataset3 As New SimpleDataset("Third", x1, y3, 0.5, 0.0)
Dim Dataset4 As New SimpleDataset("Fourth", x1, y4, 0.5, 0.0)
Dim Dataset5 As New SimpleDataset("Fifth", x1, y5, 0.5, 0.0)

pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
pTransform2.AutoScale(Dataset2, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
pTransform3.AutoScale(Dataset3, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
pTransform4.AutoScale(Dataset4, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
pTransform5.AutoScale(Dataset5, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

pTransform1.SetGraphBorderDiagonal(0.35, 0.15, 0.9, 0.65)
pTransform2.SetGraphBorderDiagonal(0.35, 0.15, 0.9, 0.65)
pTransform3.SetGraphBorderDiagonal(0.35, 0.15, 0.9, 0.65)
pTransform4.SetGraphBorderDiagonal(0.35, 0.15, 0.9, 0.65)
pTransform5.SetGraphBorderDiagonal(0.35, 0.15, 0.9, 0.65)

Dim background As _
        New Background(pTransform1, ChartObj.GRAPH_BACKGROUND, Color.White)
chartVu.AddChartObject(background)

Dim plotbackground As _
        New Background(pTransform1, ChartObj.PLOT_BACKGROUND, Color.White)
chartVu.AddChartObject(plotbackground)

Dim attrib1 As New ChartAttribute(Color.Blue, 2, DashStyle.Solid)
Dim attrib2 As New ChartAttribute(Color.Red, 2, DashStyle.Solid)
Dim attrib3 As New ChartAttribute(Color.Green, 2, DashStyle.Solid)

```

```

Dim attrib4 As New ChartAttribute(Color.Orange, 2, DashStyle.Solid)
Dim attrib5 As New ChartAttribute(Color.Magenta, 2, DashStyle.Solid)

xAxis = New LinearAxis(pTransform1, ChartObj.X_AXIS)
xAxis.SetLineWidth(2)
chartVu.AddChartObject(xAxis)

yAxis1 = New LinearAxis(pTransform1, ChartObj.Y_AXIS)
yAxis1.SetAxisIntercept(0.0)
yAxis1.SetChartObjAttributes(attrib1) ' axis color matches line color
chartVu.AddChartObject(yAxis1)

yAxis2 = New LinearAxis(pTransform2, ChartObj.Y_AXIS)
yAxis2.SetAxisIntercept(-18)
yAxis2.SetChartObjAttributes(attrib2) ' axis color matches line color
chartVu.AddChartObject(yAxis2)

yAxis3 = New LinearAxis(pTransform3, ChartObj.Y_AXIS)
yAxis3.SetAxisIntercept(-35)
yAxis3.SetChartObjAttributes(attrib3) ' axis color matches line color
chartVu.AddChartObject(yAxis3)

yAxis4 = New LinearAxis(pTransform4, ChartObj.Y_AXIS)
yAxis4.SetAxisIntercept(-52)
yAxis4.SetChartObjAttributes(attrib4) ' axis color matches line color
chartVu.AddChartObject(yAxis4)

yAxis5 = New LinearAxis(pTransform5, ChartObj.Y_AXIS)
yAxis5.SetAxisIntercept(xAxis.GetAxisMax())
yAxis5.SetAxisTickDir(ChartObj.AXIS_MAX)
yAxis5.SetChartObjAttributes(attrib5) ' axis color matches line color
chartVu.AddChartObject(yAxis5)

Dim xAxisLab As New NumericAxisLabels(xAxis)
xAxisLab.SetTextFont(theFont)
chartVu.AddChartObject(xAxisLab)

Dim yAxisLab1 As New NumericAxisLabels(yAxis1)
yAxisLab1.SetTextFont(theFont)
yAxisLab1.SetAxisLabelsFormat(ChartObj.BUSINESSFORMAT)
chartVu.AddChartObject(yAxisLab1)

Dim yAxisLab2 As New NumericAxisLabels(yAxis2)

```



```

yAxisLab2.SetTextFont(theFont)
chartVu.AddChartObject(yAxisLab2)

Dim yAxisLab3 As New NumericAxisLabels(yAxis3)
yAxisLab3.SetTextFont(theFont)
chartVu.AddChartObject(yAxisLab3)

Dim yAxisLab4 As New NumericAxisLabels(yAxis4)
yAxisLab4.SetTextFont(theFont)
chartVu.AddChartObject(yAxisLab4)

Dim yAxisLab5 As New NumericAxisLabels(yAxis5)
yAxisLab5.SetTextFont(theFont)
chartVu.AddChartObject(yAxisLab5)

```

3. Can I add new axes, text objects, plot objects, and images to a chart after it is already displayed; or must I create a new chart from scratch taking into account the additional objects?

There are two ways to add new objects to a chart. The first way is to create all objects when the chart is initially created, but disable the ones that you do not want to show up when the chart is initially rendered. Enable the objects when you want them to show up. Use the chart objects **SetChartObjEnable** method to enable/disable the object. This is useful if you are creating an animated chart where you want the chart to sequence through a predefined series of steps. The second way you add new chart objects to the **ChartView** using the **ChartView.AddChartObject** method. In both cases you need to call the **ChartView.UpdateDraw()** method after any changes are made.

The example below, extracted from the **DataCursorsAndMarkers.CustomChartDataCursor** class, creates a new **Marker** object and **NumericLabel** object each time a mouse button clicked.

[C#]

```

// create marker object at place it at the nearest point
Marker amarker = new Marker(GetChartObjScale(), MARKER_BOX,
    nearestPoint.GetX(), nearestPoint.GetY(),
    nearestPoint.GetZ(), 10.0, PHYS_POS);
amarker.SetColor(Color.White);
chartview.AddChartObject(amarker);
rNumericLabelCntr += 1.0;
// Add a numeric label the identifies the marker
pointLabel = new NumericLabel(GetChartObjScale(), textCoordsFont,

```

```

        rNumericLabelCntr, nearestPoint.GetX(), nearestPoint.GetY(),
        nearestPoint.GetZ(), PHYS_POS, DECIMALFORMAT, 0);
// Nudge text to the right and up so that it does not write over marker
pointLabel.SetTextNudge(5,-5);
chartview.AddChartObject(pointLabel);
chartview.UpdateDraw();

```

[VB]

```

Dim amarker As New Marker(GetChartObjScale(), MARKER_BOX, nearestPoint.GetX(), _
    nearestPoint.GetY(), nearestPoint.GetZ(), 10.0, PHYS_POS)
amarker.SetColor(Color.White)
chartview.AddChartObject(amarker)
rNumericLabelCntr += 1.0
' Add a numeric label the identifies the marker
pointLabel = New NumericLabel(GetChartObjScale(), textCoordsFont, _
    rNumericLabelCntr, nearestPoint.GetX(), nearestPoint.GetY(), _
    nearestPoint.GetZ(), PHYS_POS, DECIMALFORMAT, 0)
' Nudge text to the right and up so that it does not write over marker
pointLabel.SetTextNudge(5, -5)
chartview.AddChartObject(pointLabel)
chartview.UpdateDraw()

```

4. How do you zoom charts that use multiple coordinate systems?

The **ChartZoom** class will zoom one or more simultaneous coordinate systems. The example program SuperZoom zooms a chart that has one x-axis and five y-axes. Use the **ChartZoom** constructor that accepts an array of coordinate system objects.

5. How do you select a chart object and create a dialog panel that permits editing of that objects properties?

The **QCChart3D for .Net** library does not include predefined dialogs for editing chart object properties. The look, feel and details of such dialogs are application specific and it is up to the application programmer to provide these. The property editor tables common to many packages are designed to be used by developers, not end users.

You can add your own dialogs that edit the characteristics important to your end users. If you want to select the chart object by pressing a mouse button while the cursor is on the object, use the **FindObj** class. Override the **OnMouseDown** method and invoke the appropriate dialog panel there. Set the **EditChartExample** example program.

The **LineDialog** and **TextDialog** classes are derived from **System.Windows.Forms.Form** and need to be written by the programmer. Sample classes are found in the **EditChartExample** example. The sample **LineDialog** class uses the **System.Windows.Forms.ColorDialog** to select the color of line objects. The **TextDialog** class uses the **System.Windows.Forms.FontDialog** to select the text attributes associated with a font.

6. How do you handle missing data points in a chart?

There are two ways to handle missing, or bad data. The first is to mark the data point in the dataset invalid, using the datasets **SetValidData** method. The second is to set the x- and/or y- value of the bad data point to the designated bad data value, **ChartObj.rBadDataValue**. Currently this value is set equal to the value of **System.Double.MaxValue**. Either method will prevent the data point from being displayed in a chart. If the bad data value is part of a line plot, a gap will appear in the line plot at that point. Bad data points are not deleted from a dataset.

7. How do you update a chart in real-time?

In general, real-time updates involve adding new objects to a chart, or modifying existing objects that are already in the chart. Once the object is added or changed, call the **ChartView.UpdateDraw()** method to force the chart to update using the new values. Objects can be added or modified based on some external event, or in response to a timer event created using **System.Timers.Timer**. Make all changes for a given event and call the **ChartView.UpdateDraw** method once. The position of most **GraphObj** derived objects is set or modified using one of the objects **SetLocation** methods. New data points can be added to an existing dataset using one of the datasets **AddDataPoint**, **AddTimeDataPoint**, **AddGroupDataPoints** or **AddTimeGroupDataPoints** methods. **ChartPlot** derived objects that use datasets will update to reflect the new values when the **ChartView.UpdateDraw** method is called. If the coordinates of the new data points are outside of the x- and y-limits of the current coordinate system it may be necessary to rescale the coordinate system so that the new points show up; otherwise the new data points will be clipped. The new scale values can be set explicitly, or calculated using one of the auto-scale methods. The example programs **DynamicCharts.InstrumentSimulation**, **DynamicCharts.DataLogger**, **DynamicCharts.DynamicPieChart** and **DynamicCharts.MixedPlot** all demonstrate various ways to update charts in real-time.

If you want to change points in an existing dataset, but not the size of the dataset, call the datasets appropriate **SetXDataValue**, **SetYDataValue**, or **SetDataPoint** methods. The dataset has its own copy of the data so you must change these values, not the original values you used to initialize the dataset. If you plan to change every

value in the dataset, you can do that point by point, or create a new dataset and swap that in for the old dataset using the plot objects **SetDataset** or **SetGroupDataset** method. Call the **ChartView.UpdateDraw** method to force the chart to update using the new values.

8. How do I prevent flicker when updating my charts on real-time?

Flicker is the result of erasing and redrawing all or part of a chart in the current display buffer. Double buffering of screen images can minimize any flicker. The **ChartView** class does the actual work of rendering a chart image to the underlying **UserControl** display buffer. The **UserControl** class uses double buffering for the display of all screen images. When a chart is updated it is automatically rendered to an off-screen bitmap. When drawing is complete the off-screen bitmap is copied to the screen display buffer, minimizing the effect of flicker.

9. How do you implement drill down, or data tool tips in a chart?

See Chapter 16 – Data Tooltips.

Implementing drill down or tool tips consists of three major parts:

- Trapping a mouse event and determining the mouse cursor position in device and physical coordinates.
- Identifying the chart object that intersects the mouse event.
- Displaying appropriate information about the chart object.

There are many classes that aid in one or more of these functions. The **MouseListener** class will trap a mouse event in the chart view. The **FindObj** class will filter and return the chart object, if any, that intersects the mouse cursor when a mouse button is pressed. The **MoveObj** class will filter, select and move a chart object as the mouse is dragged across the chart. The **DataToolTip** class will find the data point in a chart nearest the mouse cursor and display xy information about the data point as a popup **ChartText** display. The **DataToolTip** can also be customized for the display of custom information about the selected data point. It only takes a few lines to add a simple y-value tool tip to an existing chart.

[C#]

```
DataToolTip datatooltip = new DataToolTip(chartVu);
datatooltip.SetEnable(true);
chartVu.SetCurrentMouseListener(datatooltip);
```

[Visual Basic]

```
Dim datatooltip As New DataToolTip(chartVu)
datatooltip.SetEnable(True)
chartVu.SetCurrentMouseListener(datatooltip)
```

Many of the of the example programs that include tool tips including:

Bargraphs.BarPlot3D	Bargraphs.BostonClimate
Bargraphs.DoubleBarPlot	Bargraphs.FloatingBars
Bargraphs.GroupBargraphs	Bargraphs.HistogramBars
Bargraphs.SimpleBars	CalendarData.CalendaLinePlot1
CalendarData.EcoWorld	CalendarData.EuroDollar
CalendarData.FinLogPlot	CalendarData.OpeningScreen
CalendarData.SimpleBarChart	CalendarData.TechnicalAnalysis
ContourPlots.ContourFillPlot	FinancialExamples.CandlestickChart
FinancialExamples.EuroDollar	FinancialExamples.FinLogPlot
FinancialExamples.OHLCChart	FinancialExamples.OpeningScreen
FinancialExamples.TechnicalAnalysis	FormControlExamples.ScrollablePanel
LogPlots.FinLogPlot	LogPlots.SpeakerPlot
MiscCharts.LineGap	MouseListeners.CandlestickChart
MultiLinePlots.MultiLines1	MultiLinePlots.MultiLines2
MultiLinePlots.StackedLines	MultipleAxes.MultiAxesChart
MultipleAxes.OHLCChart	PolarCharts.PolarLineAndScatterChart
ReciprocalLogPlots.ReciprocalLogPlot1	ResizeExamples.MultiLine
ResizeExamples.SimpleBarChart	ScatterPlots.ScatterPlot3D
ScatterPlots.SimpleScatter	SimpleLinePlots.LineFill
SimpleLinePlots.LinePlot3D	SimpleLinePlots.LinePlotSegments

SimpleLinePlots.PipeLineAndScatterPlot	SimpleLinePlots.Simple3DBarPlot
UserChartExample1.UserChartControl1	WebsiteStats.BandwidthUsage
WebsiteStats.SearchEngines	WebsiteStats.SearchPhrases
WebsiteStats.ViewsAndVisits	

10. I do not want to my graph to auto-scale. How do I setup the graph axes for a specific range?

Auto-scaling has two parts. The first is the auto-scaling of the coordinate system based on one or more datasets. The second part is the auto-scaling of the axes that reside in the coordinate system. Manually scale the coordinate system and axes by calling the appropriate constructors. For example:

[C#]

```

ChartCalendar xMin = new ChartCalendar(1996, ChartObj.FEBRUARY, 5);
ChartCalendar xMax = new ChartCalendar(2002, ChartObj.JANUARY, 5);
double yMin = 0;
double yMax = 105;

TimeCoordinates simpleTimeScale;
simpleTimeScale = new TimeCoordinates(xMin, yMin, xMax, yMax);
// Create the time axis (x-axis is assumed)
TimeAxis xAxis = new TimeAxis(simpleTimeScale);
// Create the linear y-axis
LinearAxis yAxis = new LinearAxis(simpleTimeScale, ChartObj.Y_AXIS);

// Create the ChartView object to place graph objects in.
ChartView chartVu = new ChartView();

// Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis);
chartVu.AddChartObject(yAxis);

```

[Visual Basic]

```

Dim xMin As ChartCalendar = New ChartCalendar(1996, ChartObj.FEBRUARY, 5)
Dim xMax As ChartCalendar = New ChartCalendar(2002, ChartObj.JANUARY, 5)
Dim yMin As Double = 0

```

```

Dim yMax As Double = 105

Dim simpleTimeScale As TimeCoordinates
simpleTimeScale = New TimeCoordinates(xMin, yMin, xMax, yMax)
' Create the time axis (x-axis is assumed)
Dim xAxis As TimeAxis = New TimeAxis(simpleTimeScale)
' Create the linear y-axis
Dim yAxis As LinearAxis = New LinearAxis(simpleTimeScale, ChartObj.Y_AXIS)

' Create the ChartView object to place graph objects in.
Dim chartVu As ChartView = New ChartView()

' Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis)
chartVu.AddChartObject(yAxis)

```

The documentation for the various coordinate system and axis classes includes examples of manual scaling.

11. How do I update my data, and auto-rescale the chart scales and axes to reflect the new data, after it has already been drawn?

Updating data was discussed in FAQ # 6. If you want the chart to rescale based on the new data, call the appropriate coordinate systems auto-scale method, followed by the auto-axis methods of all related axes. Then call the **ChartView.UpdateDraw** method. For example:

[C#]

```

// Create the ChartView object to place graph objects in.
TimeSimpleDataset Dataset1 = new TimeSimpleDataset("Sales",x1,y1);

TimeCoordinates simpleTimeCoordinates = new TimeCoordinates();
simpleTimeCoordinates.AutoScale(Dataset1,
    ChartObj.AUTOAXES_FAR , ChartObj.AUTOAXES_FAR);
ChartView chartVu = new ChartView();
// Create the time axis (x-axis is assumed)
TimeAxis xAxis = new TimeAxis(simpleTimeCoordinates);
// Create the linear y-axis
LinearAxis yAxis = new LinearAxis( simpleTimeCoordinates, ChartObj.Y_AXIS);
.
.
.

```

```
// The following code would be in the code handling the rescale event
// Rescale chart based on a modified Dataset1 dataset
simpleTimeCoordinates.AutoScale(Dataset1,
                               ChartObj.AUTOAXES_FAR , ChartObj.AUTOAXES_FAR);
xAxis.CalcAutoAxis();
yAxis.CalcAutoAxis();
// Redraw the chart using the rescaled coordinate system and axes
chartVu.UpdateDraw();
```

[Visual Basic]

```
Dim Dataset1 As TimeSimpleDataset = New TimeSimpleDataset("Sales", x1, y1)
Dim simpleTimeCoordinates As TimeCoordinates = New TimeCoordinates()
simpleTimeCoordinates.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, _
                               ChartObj.AUTOAXES_FAR)
Dim chartVu As ChartView = New ChartView()
' Create the time axis (x-axis is assumed)
Dim xAxis As TimeAxis = New TimeAxis(simpleTimeCoordinates)
' Create the linear y-axis
Dim yAxis As LinearAxis = New LinearAxis(simpleTimeCoordinates, ChartObj.Y_AXIS)

' The following code would be in the code handling the rescale event
' Rescale chart based on a modified Dataset1 dataset
simpleTimeCoordinates.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, _
                               ChartObj.AUTOAXES_FAR)
xAxis.CalcAutoAxis()
yAxis.CalcAutoAxis()
' Redraw the chart using the rescaled coordinate system and axes
chartVu.UpdateDraw()
```

12. When I use the auto-scale and auto-axis routines my semi-log chart has the log axis scaled using powers of 10 (1, 10,100, 1000, etc.) as the starting and ending values, or as the major tick interval for labeling. How do I make my log graphs start at 20 and end at 50,000, with major tick marks at 20, 200, 2000 and 20000?

The auto-scale routines for logarithmic coordinate systems will always select a power of 10 for the minimum and maximum value of the scale. You can use the auto-scale routine and then override the minimum and/or maximum values for the logarithmic scale. The default **LogAxis** constructor will pick up on the minimum of the coordinate system and use that as the axis tick mark origin. Or you can leave the coordinate system unchanged, and change the starting point of the axis tick marks

using the axis **SetAxisTickOrigin** method. The example below is derived from the Logarithmic example code.

[C#]

```
GroupDataset Dataset1 = new GroupDataset("First",x1,y1);

CartesianCoordinates pTransform1 = new CartesianCoordinates(ChartObj.LOG_SCALE,
                                                             ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);

pTransform1.SetScaleStartX(20); // Force start of scale at 20, AutoScale will
                               // always choose a power of 10 decade.
LogAxis xAxis = new LogAxis(pTransform1, ChartObj.X_AXIS);
xAxis.SetAxisTickOrigin(20);
chartVu.AddChartObject(xAxis);
```

[Visual Basic]

```
' Create the ChartView object to place graph objects in.
Dim Dataset1 As GroupDataset = New GroupDataset("First", x1, y1)

Dim pTransform1 As CartesianCoordinates = _
    New CartesianCoordinates(ChartObj.LOG_SCALE, ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

pTransform1.SetScaleStartX(20) ' Force start of scale at 20, AutoScale will
' always choose a power of 10 decade.
Dim xAxis As LogAxis = New LogAxis(pTransform1, ChartObj.X_AXIS)
xAxis.SetAxisTickOrigin(20)
chartVu.AddChartObject(xAxis)
```

13. How do I create and use custom, multi-line string labels as the axis labels for my graph?

The **StringAxisLabels** class should be used to create multi-line axis labels. Insert the “\n” new line character to add additional lines to each string used to define the string axis labels. The example below is from the AxisLabels example program.

[C#]

```
String []xstringlabels =
```

```

        {
            "",
            "Western" + "\n" + "Sales" + "\n" + "Region",
            "Eastern" + "\n" + "Sales" + "\n" + "Region",
            "Southern" + "\n" + "Sales" + "\n" + "Region",
            "Northern" + "\n" + "Sales" + "\n" + "Region"};

        StringAxisLabels xAxisLab5 = new StringAxisLabels(xAxis5);
        xAxisLab5.SetAxisLabelsStrings(xstringlabels, 5);
        xAxisLab5.SetTextFont(graph5Font);
        chartVu.AddChartObject(xAxisLab5);

```

[Visual Basic]

```

Dim xstringlabels As [String]() = {"", "Western" + ControlChars.Lf + "Sales" + _
    ControlChars.Lf + "Region", "Eastern" + ControlChars.Lf + "Sales" + _
    ControlChars.Lf + "Region", "Southern" + ControlChars.Lf + "Sales" + _
    ControlChars.Lf + "Region", "Northern" + ControlChars.Lf + "Sales" + _
    ControlChars.Lf + "Region"}

Dim xAxisLab5 As New StringAxisLabels(xAxis5)
xAxisLab5.SetAxisLabelsStrings(xstringlabels, 5)
xAxisLab5.SetTextFont(graph5Font)
chartVu.AddChartObject(xAxisLab5)

```

14. How do I place more than one graph in a view?

One way to create multiple charts is to create multiple instances of the **ChartView** class and add each **ChartView** object to a container object such as a **UserControl**. A layout manager manages the position and size of each **ChartView**. Another way is to place multiple charts in the same **ChartView** object. This makes it easier to guarantee alignment between the axes of separate graphs. The trick to doing this is to create separate coordinate system objects (**CartesianCoordinates**, **TimeCoordinates**) for each chart, and to position the plot area volume of each coordinate system so that they do not overlap. Use one of the coordinate systems **SetGraphBorder...** methods. Many of the examples use this technique, including:

Bargraphs.BostonClimate

Bargraphs.DoubleBarPlot

Bargraphs.FloatingBars

Bargraphs.GroupBargraphs

CalendarData.OpeningScreen

FinancialExamples.OHLCChart

FinancialExamples.OpeningScreen

[C#]

```
pTransform1 = new TimeCoordinates();
pTransform1.SetGraphBorderDiagonal(0.1, .15, .90, 0.6) ;

pTransform2 = new TimeCoordinates();
pTransform2.SetGraphBorderDiagonal(0.1, .7, .90, 0.875) ;
```

[Visual Basic]

```
pTransform1 = new TimeCoordinates()
pTransform1.SetGraphBorderDiagonal(0.1, .15, .90, 0.6)

pTransform2 = new TimeCoordinates()
pTransform2.SetGraphBorderDiagonal(0.1, .7, .90, 0.875)
```

15. How do I use your software to generate GIF files?

Unlike the JPEG image file format, the GIF file format uses a proprietary data compression algorithm known as LZW. The patent on the LZW compression algorithm is owned by the large computer/data processing company Unisys. Programmers who write commercial applications that use this file format may be subject to paying Unisys royalties. For this reason we do not intent to write any code that explicitly supports the GIF file format. If you, after having considered all of the alternatives, still require GIF files then create a

System.Drawing.Imaging.ImageFormat instance that uses a GIF image format and use that to initialize the **chart3dnet.BufferedImage** class. The same holds true for all image formats

16. Sometimes the major tick marks of an axis are missing the associated tick mark label ?

The axis labeling routines are quite intelligent. Before the label is drawn at its calculated position, the software does a check to see if the bounding box of the new axis label intersects the bounding box of the previous axis label. If the new label is

going to overlap the previous label, the label is skipped. You can override this default behavior by calling the objects **SetOverlapLabelMode** method.

```
SetOverlapLabelMode (ChartObj.OVERLAP_LABEL_DRAW);
```

Another option, for horizontal axes only, is to stagger the tick mark labels. A stagger automatically alternates the line on which the tick mark label is placed.

```
SetOverlapLabelMode (ChartObj.OVERLAP_LABEL_STAGGER);
```

17. How do I change the order the chart objects are drawn? For example, I want one of my grid objects to be drawn under the charts line plot objects, and another grid object to be drawn top of the charts line plot objects.

There are two ordering methods used to render chart objects. The first method renders the objects in order, as added to the **ChartView** object. Objects added to the view last are drawn on top of objects added first. The second method renders the objects according to their z-order. Objects with the lowest z-order values are rendered first. Objects with equal z-order values are rendered in the order they are added to the **ChartView** object. The second method (z-order rendering) is the default method of object rendering used by the **ChartView** class. This default behavior can be changed by call the **ChartView.SetZOrderSortEnable(false)** method.

You can change the default z-order value on an object-by-object basis. Call the **GraphObj.SetZOrder** method to change the z-order for any given object.

See the section in the manual titled *Rendering Order of GraphObj Objects* for information about the default z-values for all chart objects

The example below sets the z-order value of grid1 to something less than the default value (50) of **ChartPlot** objects, and the z-order value of grid2 to something greater than the default value.

[C#]

```
ChartView chartVu = new ChartView();
.
.
.

Grid grid1 = new Grid(xAxis, yAxis, ChartObj.Y_AXIS, ChartObj.GRID_MAJOR);
grid1.SetZOrder(40); // This is actually the default value for the grid z-order
chartVu.AddChartObject(grid1);

Grid grid2 = new Grid(xAxis, yAxis, ChartObj.Y_AXIS, ChartObj.GRID_MINOR);
```

```
grid2.SetZOrder(150); // Grid is drawn after ChartPlot objects
                        // which have default z-value of 50
chartVu.AddChartObject(grid2);
```

[Visual Basic]

```
Dim chartVu As ChartView = new ChartView()
.
.
.

Dim grid1 As Grid = new Grid(xAxis, yAxis, ChartObj.Y_AXIS, ChartObj.GRID_MAJOR)
grid1.SetZOrder(40) ` This is actually the default value for the grid z-order
chartVu.AddChartObject(grid1)

Dim grid2 As Grid = new Grid(xAxis, yAxis, ChartObj.Y_AXIS, ChartObj.GRID_MINOR)
grid2.SetZOrder(150) ` Grid is drawn after ChartPlot objects
                     ` which have default z-value of 50
chartVu.AddChartObject(grid2)
```

18. How to I use a ScrollBar object to control horizontal scrolling of the data in my chart?

Since the **ChartView** class is derived from **UserControl**, you can place the scroll bar in the **ChartView** object, or you can place the **ChartView** object and the scroll bar in a parent container and use a layout manager to position everything. If you place the scroll bar in the **ChartView** you can still position it using a layout manager. The **ChartView** will always use the entire content area of the underlying **UserControl** for its canvas and the scroll bars will sit on top of this, not side by side. The example program `FormControlExamples.LinePlotScrollBar` uses two scroll bars, a horizontal scroll bar to control scrolling of the x-axis, and a vertical scroll bar that controls the magnitude of the y-axis. You need to add `hScrollBar1_Scroll` and `vScrollBar1_Scroll` event listeners to the **ChartView** class to process changes in scroll bar values.

[C#]

```
public void UpdateXScaleAndAxes(int index)
{
    int startindex = index;
    pTransform1.SetScaleStartX( (double) startindex);
```

```

        pTransform1.SetScaleStopX( (double) (startindex + 100));
        xAxis.CalcAutoAxis();
        yAxis.CalcAutoAxis();
        xAxisLab.CalcAutoAxisLabels();
        yAxisLab.CalcAutoAxisLabels();
        this.UpdateDraw();
    }

    public void UpdateYScaleAndAxes(int index)
    {
        int startindex = index;
        pTransform1.SetScaleStartY( (double) -startindex);
        pTransform1.SetScaleStopY( (double) startindex);
        xAxis.CalcAutoAxis();
        yAxis.CalcAutoAxis();
        xAxisLab.CalcAutoAxisLabels();
        yAxisLab.CalcAutoAxisLabels();
        this.UpdateDraw();
    }

    private void hScrollBar1_Scroll(object sender,
                                     System.Windows.Forms.ScrollEventArgs e)
    {
        UpdateXScaleAndAxes(hScrollBar1.Value);
    }

    private void vScrollBar1_Scroll(object sender,
                                     System.Windows.Forms.ScrollEventArgs e)
    {
        UpdateYScaleAndAxes(vScrollBar1.Value);
    }

```

[Visual Basic]

```

Public Sub UpdateXScaleAndAxes(ByVal index As Integer)
    Dim startindex As Integer = index
    pTransform1.SetScaleStartX(CDbl(startindex))
    pTransform1.SetScaleStopX(CDbl(startindex + 100))
    xAxis.CalcAutoAxis()
    yAxis.CalcAutoAxis()
    xAxisLab.CalcAutoAxisLabels()
    yAxisLab.CalcAutoAxisLabels()

```

```

        Me.UpdateDraw()
    End Sub 'UpdateXScaleAndAxes

Public Sub UpdateYScaleAndAxes(ByVal index As Integer)
    Dim startindex As Integer = index
    pTransform1.SetScaleStartY(CDbl(-startindex))
    pTransform1.SetScaleStopY(CDbl(startindex))
    xAxis.CalcAutoAxis()
    yAxis.CalcAutoAxis()
    xAxisLab.CalcAutoAxisLabels()
    yAxisLab.CalcAutoAxisLabels()
    Me.UpdateDraw()
End Sub 'UpdateYScaleAndAxes

Private Sub HScrollBar2_Scroll(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.ScrollEventArgs)
    If Not (Me.IsDesignMode) Then
        UpdateXScaleAndAxes(HScrollBar1.Value)
    End If
End Sub

Private Sub VScrollBar2_Scroll(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.ScrollEventArgs)
    If Not (Me.IsDesignMode) Then
        UpdateYScaleAndAxes(VScrollBar1.Value)
    End If
End Sub

```

There are many other examples of Form components interacting with charts. The ContourPlots.ContourLinePlot example program uses a **CheckBox** object to specify which contours are to be displayed. The FormControlExamples.CandlestickWithTable example continuously updates a **DataGrid** (the base class for the MyDataGrid class) table with open-high-low-close data and simultaneously updates a chart with the same data. The scrolling action of the **DataGrid** is synchronized with the chart. In the FormControlExamples.OHLCChart example a **Scrollbar** controls the time axis of a stock market OHLC chart. The MultipleAxes.MultiAxesChart example uses **Button** objects to select the x-axis range.

19. I am trying to plot 100,000,000 data points and it takes too long to draw the graph. What is wrong with the software and what can I do to make it faster?

The software runs as fast as we can make it. We do not have any hidden switches that will speed up the software. What you need to do is to step back and think about the best way to display your data.

A fundamental issue that many programmers fail to consider is the relationship between the resolution of the rasterized screen image of the plot and the resolution of the data. A typical chart image will have 500-1000 pixels as the horizontal resolution of the plotting area. This would imply that in the 100M data point example above, every horizontal pixel would represent 50K to 100K data points. Obviously this is a terrible mismatch. In fact it is a bad match for datasets that have more than a couple of thousands points.

So what you do is compress the data before it is displayed. Take the 100M data points and compress them down to 2K data points. The data compression can take several forms. You can take an average of every N points. The resulting dataset will be reduced by a factor of N. You can also find the sum for every N points, the minimum value of every N points, the maximum of every N points, or both the minimum and maximum of every 2N points. The last compression method, minimum and maximum, will always capture any minimums and maximum in the data. The result is that a 2000 point compressed dataset, where there are at least two data points per pixel of horizontal resolution, will look just like the 100,000,000 point dataset, only display hundreds of times faster. The **Datset** classes all include compression methods (**SimpleDataset.CompressSimpleDataset**, **GroupDataset.CompressGroupDataset**, **TimeSimpleDataset.CompressTimeSimpleDataset** and **TimeGroupDataset.CompressTimeGroupDataset**, **TimeGroupDataset.CompressTimeFieldSimpleDataset**, **TimeGroupDataset.CompressTimeFieldGroupDataset**) that operate on the existing dataset and return a new, compressed dataset. The **CompressTimeFieldSimpleData** and **CompressTimeFieldGroupDataset** are particular useful because they do not use a fixed sample size of N, instead they compress data so that adjacent time values are an increment of a specific time field (**ChartObj.DAY_OF_YEAR**, **ChartObj.WEEK_OF_YEAR**, **ChartObj.MONTH**, **ChartObj.Year**). Compressing data by month and year obviously requires a varying sample size.

Once created, connect the compressed dataset to the **ChartPlot** object used to display the dataset.

[C#]

```
nNumPnts = 1000000;
TimeSimpleDataset RawDataset = new
    TimeSimpleDataset("Raw", xtimedata, ydata, nNumPnts);
```



```

int compressXmode = ChartObj.DATACOMRESS_AVERAGE;
int compressYmode = ChartObj.DATACOMRESS_MINMAX;
int compressTimeField = Calendar.MONTH;
TimeSimpleDataset CompressedDataset =
    RawDataset.CompressTimeFileSimpleData( compressXmode,
                                            compressYmode,
                                            compressTimeField,
                                            0, nNumPnts, "Compressed");

```

[Visual Basic]

```

nNumPnts = 1000000
Dim RawDataset As TimeSimpleDataset = new _
    TimeSimpleDataset("Raw", xtimedata, ydata, nNumPnts)
Dim compressXmode As Integer = ChartObj.DATACOMRESS_AVERAGE
Dim compressYmode As Integer = ChartObj.DATACOMRESS_MINMAX
Dim compressTimeField As Integer = Calendar.MONTH
Dim CompressedDataset As TimeSimpleDataset = _
    RawDataset.CompressTimeFileSimpleData( compressXmode,
                                            compressYmode,
                                            compressTimeField,
                                            0, nNumPnts, "Compressed")

```

20. How do I get data from my database into a chart?

The real question is: How do you get data from your database into a simple .Net program, storing sequential data values in data array variables. This is up to you and is independent of the charting software. We recommend that you use the SQL database classes that are part of .Net and study the documentation provide by Microsoft and other sources, such as the O'Reilly programming books. Once you can read individual data elements of your data base it is a trivial matter to place the numeric and calendar data into simple .Net array variables and from there plot the data.

21. How do I use this charting software to generate chart images “on-the-fly” from a server?

The **BufferedImage** class creates chart images independent of any physical display context. The **BufferedImage** class uses a **System.Drawing.Imaging.ImageFormat** object to control the format of the resulting image bitmap. You can use this image as an image object in an HTML page. See the previous chapter on ASP.Net programming for examples.

22. Can QCChart3D for .Net be used to create programs and controls that run like Java applets in web browsers?

Yes, streaming images created using QCChart3D to a web site, and embedding a QCChart3D.ChartView user control in a web page is discussed in Chapter 24 – Using QCChart3D for .Net to Create Web Applications.

23. Are you going to add additional real-time routines to the QCChart3D for .Net library?

You should probably be using our the **Real-Time Graphics Tools for .Net** which uses our QCChart2D for .Net software as its base.

<http://www.quinn-curtis.com/QCRTGraphProdPage.htm>

INDEX

- 3D Points, 55, 58, 59, 61, 66, 68, 69, 72, 81, 82, 83, 84, 85, 115, 116, 147, 148, 155, 156, 165, 221, 222, 223, 224, 226, 227, 230, 238, 239, 243, 244, 250, 251, 257, 258, 261, 262, 268, 269, 270, 272, 274, 275, 278, 280, 283, 284, 291, 292, 297, 304, 308, 364, 366
- 3D Text, 50, 61, 149, 194, 195, 200, 206, 330, 331, 351, 354, 355, 381, 382, 383, 384, 385, 386, 387, 389, 390, 391, 392, 395, 398, 472
- 3D Walls, 21, 22, 61, 149, 157, 163, 164, 165, 166
- Arrow, 30, 61, 248, 249, 250, 251, 401, 408, 409
- Arrow plots, 29, 30, 61, 247, 248, 249, 250, 251, 255, 401, 408
- ArrowPlot, 29, 30, 61, 247, 248, 249, 250, 251, 255, 401, 408
- Arrows, 30, 61, 248, 249, 250, 251, 401, 408, 409
- ASP.NET, 9
- Auto-scaling classes, 19, 61, 115, 120, 121, 122, 123, 125, 130, 131, 132, 133, 134, 136, 139, 140, 142, 144, 147, 162, 165, 221, 222, 223, 224, 226, 227, 230, 233, 234, 238, 239, 243, 250, 251, 254, 257, 258, 261, 262, 270, 271, 274, 275, 278, 279, 287, 288, 291, 292, 297, 303, 306, 307, 323, 324, 331, 332, 334, 335, 344, 345, 434, 442, 464, 467, 476, 477, 478
- Axis, 22, 23, 26, 50, 61, 149, 167, 168, 169, 170, 175, 176, 182, 183, 193, 194, 195, 196, 200, 206, 211, 330, 390
- Axis label classes, 26, 27, 50, 61, 149, 193, 194, 195, 200, 206, 381, 392, 478
- Axis titles, 50, 61, 216, 330, 381, 390, 391, 392
- AxisLabels, 26, 27, 61, 149, 193, 194, 195, 200, 206, 381, 478
- AxisTitle, 50, 61, 216, 330, 381, 390, 391, 392
- Background, 1, 20, 21, 61, 147, 148, 149, 157, 158, 160, 161, 162, 163, 165, 166, 267, 269, 270, 271, 306, 365, 367, 434, 442, 465, 467
- Backgrounds, 1, 20, 21, 61, 147, 148, 149, 157, 158, 160, 161, 162, 163, 165, 166, 267, 269, 270, 271, 306, 365, 367, 434, 442, 465, 467
- Bar plots, 42, 43, 61, 219, 231, 232, 233, 234, 235, 236, 329, 349
- BarDatapointValue, 61
- Bubble plot legend items, 48, 49, 61, 376
- Bubble plot legends, 48, 49, 61, 371, 376, 377, 378, 379, 380
- Bubble plots, 29, 32, 61, 247, 252, 253, 254, 376, 377, 378, 379
- BubblePlot, 29, 32, 61, 247, 252, 253, 254, 376, 377, 378, 379
- BubblePlotLegend, 48, 49, 61, 371, 376, 377, 378, 379, 380
- BubblePlotLegendItem, 48, 49, 61, 376
- Buffered images, 54, 61, 413, 421, 422, 423, 431, 433, 435, 436, 441, 443, 480, 486
- BufferedImage, 54, 61, 413, 421, 422, 423, 431, 433, 435, 436, 441, 443, 480, 486
- Calendar utilities, 15, 16, 19, 26, 50, 54, 55, 61, 63, 73, 74, 75, 76, 77, 78, 79, 80, 81, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 141, 147, 148, 182, 187, 189, 190, 191, 204, 233, 234, 235, 236, 258, 259, 266, 270, 271, 274, 275, 277, 278, 279, 280, 287, 288, 297, 298, 300, 301, 303, 304, 346, 395, 396, 451, 456, 475
- Candlestick plots, 29, 32, 61, 247, 255, 256, 258, 350
- CandlestickPlot, 29, 32, 61, 247, 255, 256, 258, 350
- Cartesian coordinates, 17, 18, 58, 61, 105, 108, 114, 117, 118, 119, 120, 121, 122, 123, 124, 125, 127, 140, 141, 142, 144, 152, 153, 164, 165, 173, 174, 180, 198, 199, 223, 224, 230, 243, 250, 251, 261, 267, 269, 282, 284, 291, 292, 306, 307, 316, 318, 323, 324, 334, 335, 339, 340, 343, 344, 345, 346, 362, 364, 366, 434, 442, 464, 466, 467, 477, 478, 479
- CartesianCoordinates, 17, 18, 58, 61, 105, 108, 114, 117, 118, 119, 120, 121, 122, 123, 124, 125, 127, 140, 141, 142, 144, 152, 153, 164, 165, 173, 174, 180, 198, 199, 223, 224, 230, 243, 250, 251, 261, 267, 269, 282, 284, 291, 292, 306, 307, 316, 318, 323, 324, 334, 335, 339, 340, 343, 344, 345, 346, 362, 364, 366, 434, 442, 464, 466, 467, 477, 478, 479
- Cell plots, 29, 33, 61, 247, 255, 259, 260, 261, 262
- CellPlot, 29, 33, 61, 247, 255, 259, 260, 261, 262
- Chart depth, 59, 60, 115, 145, 146, 147, 148, 165, 221, 222, 227, 230, 235, 239, 243, 244, 250, 251, 257, 258, 261, 262, 268, 269, 270, 272, 274, 275, 278, 280, 291, 292, 297, 298, 304, 308
- Chart object attributes, 13, 18
- Chart titles, 50, 61, 330, 381, 387, 388, 389, 390, 391, 435, 442, 451, 455
- ChartAttribute, 18, 19, 20, 61, 162, 163, 164, 165, 166, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 242, 243, 244, 249, 251, 253, 254, 256, 257, 258, 260, 261, 262, 263, 265, 268, 269, 270, 272, 273, 274, 275, 276, 277, 278, 280, 281, 283, 284, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 298, 299, 300, 302, 305, 306, 307, 308, 309, 310, 311, 315, 317, 318, 354, 361, 362, 364, 365, 366, 372, 373, 374, 375, 376, 377, 378, 379, 380, 403, 404, 409, 435, 442, 465, 467
- ChartCalendar, 15, 16, 19, 26, 50, 54, 55, 61, 63, 73, 74, 75, 76, 77, 78, 79, 80, 81, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 141, 147, 148, 182, 187, 189, 190, 191, 204, 233, 234, 235, 236, 258, 259, 266, 270, 271, 274, 275, 277, 278, 279, 280, 287, 288, 297, 298, 300, 301, 303, 304, 346, 395, 396, 451, 456, 475
- ChartImage, 51, 61, 161, 401, 405, 406, 407, 408, 441, 443
- ChartLabel, 50, 61, 381, 392

- ChartObj, 61, 64, 69, 72, 75, 76, 77, 80, 81, 87, 90, 94, 96, 97, 99, 102, 103, 115, 121, 123, 124, 125, 128, 129, 130, 131, 132, 133, 134, 136, 137, 138, 139, 140, 141, 144, 147, 148, 150, 151, 158, 162, 163, 164, 165, 166, 173, 174, 175, 180, 181, 182, 189, 190, 191, 198, 199, 200, 204, 205, 206, 208, 213, 214, 215, 216, 217, 221, 222, 223, 224, 226, 227, 230, 233, 234, 235, 236, 238, 239, 240, 243, 244, 245, 250, 251, 254, 255, 257, 258, 261, 262, 267, 268, 269, 270, 271, 272, 274, 275, 277, 278, 279, 280, 283, 285, 287, 288, 290, 291, 292, 297, 298, 300, 301, 303, 304, 306, 307, 317, 319, 323, 324, 326, 327, 330, 331, 332, 334, 335, 340, 341, 344, 345, 346, 347, 351, 353, 354, 357, 358, 359, 365, 367, 375, 376, 379, 380, 385, 386, 387, 389, 390, 391, 398, 399, 404, 407, 408, 409, 410, 413, 421, 434, 435, 442, 456, 463, 464, 465, 466, 467, 468, 471, 475, 476, 477, 478, 480, 481, 482, 485, 486
- ChartPlot, 28, 29, 61, 63, 149, 150, 156, 219, 225, 228, 231, 236, 241, 248, 252, 255, 259, 263, 264, 272, 275, 280, 285, 289, 293, 295, 298, 301, 304, 308, 313, 355, 357, 358, 361, 373, 374, 381, 471, 481, 482, 485
- ChartPrint, 54, 61, 413, 414, 415, 416, 417, 418, 419, 420
- ChartScale, 17, 18, 61, 105, 107
- ChartShape, 51, 61, 401, 402, 403, 404, 408, 409, 410
- ChartSymbol, 51, 61, 351, 354
- ChartText, 50, 61, 149, 194, 195, 200, 206, 330, 331, 351, 354, 355, 381, 382, 383, 384, 385, 386, 387, 389, 390, 391, 392, 395, 398, 472
- ChartTitle, 50, 61, 330, 381, 387, 388, 389, 390, 391, 435, 442, 451, 455
- ChartView, 6, 8, 9, 10, 12, 14, 20, 52, 54, 59, 61, 66, 72, 75, 80, 89, 93, 96, 102, 143, 144, 145, 147, 149, 150, 151, 152, 153, 154, 155, 157, 161, 173, 174, 180, 181, 190, 191, 198, 199, 204, 205, 325, 329, 330, 331, 332, 333, 334, 338, 339, 340, 341, 342, 343, 344, 350, 352, 356, 363, 365, 371, 381, 401, 403, 413, 414, 415, 416, 417, 418, 420, 421, 422, 423, 425, 426, 428, 429, 431, 432, 433, 434, 435, 436, 440, 441, 443, 447, 448, 449, 450, 451, 453, 455, 464, 466, 469, 471, 472, 475, 476, 477, 478, 479, 481, 482, 486
- Comma separated values, 54, 55, 61, 63, 64, 65, 66, 68, 70, 71, 74, 76, 77, 79, 83, 84, 87, 89, 90, 91, 92, 93, 95, 96, 98, 100, 101, 248
- Contour plotting, 14, 15, 61, 63, 64, 81, 82, 83, 85, 86, 87, 313, 315, 334, 335
- ContourDataset, 14, 15, 61, 63, 64, 81, 82, 83, 85, 86, 87, 313, 315, 334, 335
- CSV, 54, 55, 61, 63, 64, 65, 66, 68, 70, 71, 74, 76, 77, 79, 83, 84, 87, 89, 90, 91, 92, 93, 95, 96, 98, 100, 101, 248
- Data compression, 485
- Data cursors, 52, 61, 321, 324, 325, 326, 327
- DataCursor, 52, 61, 321, 324, 325, 326, 327
- Dataset, 14, 15, 28, 58, 61, 63, 64, 69, 72, 77, 81, 86, 87, 90, 94, 99, 109
- Dataset classes, 14, 15, 28, 58, 61, 63, 64, 69, 72, 77, 81, 86, 87, 90, 94, 99, 109
- DataToolTip, 52, 53, 61, 349, 350, 351, 352, 353, 354, 355, 356, 472
- Dimension, 54, 55, 58, 61, 340, 341, 346, 347, 407, 408
- Error bar plots, 29, 33, 61, 247, 263, 264
- ErrorBarPlot, 29, 33, 61, 247, 263, 264
- Finding graph objects, 52, 61, 155, 156, 470, 472
- FindObj, 52, 61, 155, 156, 470, 472
- Floating bar plots, 29, 34, 61, 247, 264, 265, 268, 269, 270, 272, 273
- Floating stacked bar plots, 35, 272, 273
- FloatingBarPlot, 29, 34, 61, 247, 264, 265, 268, 269, 270, 272, 273
- FloatingStackedBarPlot, 35, 272, 273
- Graph object class, 19, 20, 61, 108, 143, 149, 150, 151, 153, 154, 155, 156, 157, 163, 168, 169, 175, 182, 194, 195, 200, 206, 211, 212, 219, 225, 228, 231, 236, 241, 248, 252, 255, 259, 263, 264, 272, 275, 280, 285, 289, 293, 295, 298, 301, 304, 308, 313, 321, 324, 329, 330, 332, 337, 361, 371, 373, 374, 376, 382, 402, 405, 471, 481
- GraphObj, 19, 20, 61, 108, 143, 149, 150, 151, 153, 154, 155, 156, 157, 163, 168, 169, 175, 182, 194, 195, 200, 206, 211, 212, 219, 225, 228, 231, 236, 241, 248, 252, 255, 259, 263, 264, 272, 275, 280, 285, 289, 293, 295, 298, 301, 304, 308, 313, 321, 324, 329, 330, 332, 337, 361, 371, 373, 374, 376, 382, 402, 405, 471, 481
- Grid, 49, 61, 149, 150, 211, 213, 214, 216, 217, 481, 482
- Grids, 49, 61, 149, 150, 211, 213, 214, 216, 217, 481, 482
- Group 3D datasets, 14, 16, 19, 61, 63, 64, 90, 91, 92, 93
- Group bar plots, 29, 35, 61, 275, 276, 278, 280, 350
- Group datasets, 14, 16, 19, 58, 61, 63, 87, 88, 89, 90, 91, 92, 94, 99, 242, 249, 250, 251, 253, 255, 256, 260, 261, 262, 263, 265, 267, 269, 273, 276, 281, 282, 284, 286, 289, 291, 292, 293, 296, 299, 302, 305, 306, 307, 309, 477, 478, 485
- Group plot classes, 12, 28, 29, 61, 81, 149, 247, 248, 252, 255, 259, 263, 264, 272, 275, 276, 280, 285, 289, 293, 295, 298, 301, 304, 305, 308
- GroupBarPlot, 29, 35, 61, 275, 276, 278, 280, 350
- GroupDataset, 14, 16, 19, 58, 61, 63, 87, 88, 89, 90, 91, 92, 94, 99, 242, 249, 250, 251, 253, 255, 256, 260, 261, 262, 263, 265, 267, 269, 273, 276, 281, 282, 284, 286, 289, 291, 292, 293, 296, 299, 302, 305, 306, 307, 309, 477, 478, 485
- GroupDataset3D, 14, 16, 19, 61, 63, 64, 90, 91, 92, 93
- GroupPlot, 12, 28, 29, 61, 81, 149, 247, 248, 252, 255, 259, 263, 264, 272, 275, 276, 280, 285, 289, 293, 295, 298, 301, 304, 305, 308
- Histogram plots, 29, 36, 61, 247, 280, 281, 283, 284
- HistogramPlot, 29, 36, 61, 247, 280, 281, 283, 284
- Image objects, 51, 61, 161, 401, 405, 406, 407, 408, 441, 443
- Legend, 48, 49, 61, 150, 371, 381
- Legend classes, 48, 49, 61, 150, 371, 381
- Legend items, 48, 49, 61, 373, 374
- LegendItem, 48, 49, 61, 373, 374
- Line gap plots, 29, 37, 61, 247, 285, 286, 287, 288

- Line marker plots, 42, 44, 61, 219, 241, 242, 243, 244, 329
- Line plots, 42, 46, 61, 219, 220, 221, 222, 223, 224, 228, 327, 329, 349, 435, 442
- Linear auto-scaling, 19, 61
- Linear axis, 22, 24, 27, 49, 61, 144, 150, 151, 167, 168, 169, 170, 171, 173, 174, 175, 180, 182, 189, 190, 191, 193, 195, 198, 199, 204, 205, 211, 213, 215, 391, 434, 442, 451, 455, 463, 465, 466, 467, 468, 475, 476, 477
- Linear scale, 17, 61, 105, 108
- LinearAxis, 22, 24, 27, 49, 61, 144, 150, 151, 167, 168, 169, 170, 171, 173, 174, 175, 180, 182, 189, 190, 191, 193, 195, 198, 199, 204, 205, 211, 213, 215, 391, 434, 442, 451, 455, 463, 465, 466, 467, 468, 475, 476, 477
- LinearScale, 17, 61, 105, 108
- LineGapPlot, 29, 37, 61, 247, 285, 286, 287, 288
- Log scale, 17, 61, 105, 108
- Logarithmic auto-scaling, 19, 61
- Logarithmic axis, 22, 24, 27, 49, 61, 167, 168, 175, 177, 178, 180, 193, 195, 211, 477, 478
- LogAxis, 22, 24, 27, 49, 61, 167, 168, 175, 177, 178, 180, 193, 195, 211, 477, 478
- LogScale, 17, 61, 105, 108
- Marker, 51, 52, 61, 241, 321, 322, 323, 324, 325, 327, 469, 470
- Markers, 51, 52, 61, 241, 321, 322, 323, 324, 325, 327, 469, 470
- MouseListeners, 52, 61, 325, 329, 330, 332, 333, 349, 350, 472
- MoveData, 52, 53, 61, 329, 332, 333, 334, 335
- MoveObj, 52, 61, 329, 330, 332, 472
- Moving chart data, 52, 53, 61, 329, 332, 333, 334, 335
- Moving graph objects, 52, 61, 329, 330, 332, 472
- Multi-line plots, 29, 61, 247, 289, 290, 292, 294, 295, 350
- MultiLinePlot, 29, 61, 247, 289, 290, 292, 294, 295, 350
- Multi-ribbon plots, 29, 38, 291, 293, 294, 295
- MultiRibbonPlot, 29, 38, 291, 293, 294, 295
- Nearest point class, 55, 56, 61
- NearestPointData, 55, 56, 61
- Numeric axis labels, 26, 27, 61, 193, 195, 196, 198, 199, 204, 205, 215, 391, 434, 442, 466, 468, 469
- Numeric data point labels, 61
- Numeric labels, 50, 61, 236, 244, 283, 285, 303, 304, 317, 319, 349, 351, 353, 354, 362, 365, 367, 381, 392, 393, 394, 398, 399, 469, 470
- NumericAxisLabels, 26, 27, 61, 193, 195, 196, 198, 199, 204, 205, 215, 391, 434, 442, 466, 468, 469
- NumericLabel, 50, 61, 236, 244, 283, 285, 303, 304, 317, 319, 349, 351, 353, 354, 362, 365, 367, 381, 392, 393, 394, 398, 399, 469, 470
- OHLCBarPlot, 29, 40, 61, 247, 298, 299, 300, 301
- OHLCPlot, 29, 39, 61, 229, 247, 295, 296, 297, 298, 299, 350
- Open-High-Low-Close bar plots, 29, 40, 61, 247, 298, 299, 300, 301
- Open-High-Low-Close plots, 29, 39, 61, 229, 247, 295, 296, 297, 298, 299, 350
- Physical coordinates, 17, 18, 20, 22, 61, 105, 107, 108, 117, 128, 157, 160, 161, 163, 167, 169, 171, 175, 177, 220, 225, 228, 229, 232, 237, 242, 249, 253, 256, 260, 263, 265, 273, 276, 281, 286, 289, 293, 296, 299, 302, 305, 309, 315, 321, 322, 325, 333, 338, 341, 342, 357, 358, 361, 362, 383, 384, 388, 392, 393, 394, 395, 396, 397, 402, 405, 406
- PhysicalCoordinates, 17, 18, 20, 22, 61, 105, 107, 108, 117, 128, 157, 160, 161, 163, 167, 169, 171, 175, 177, 220, 225, 228, 229, 232, 237, 242, 249, 253, 256, 260, 263, 265, 273, 276, 281, 286, 289, 293, 296, 299, 302, 305, 309, 315, 321, 322, 325, 333, 338, 341, 342, 357, 358, 361, 362, 383, 384, 388, 392, 393, 394, 395, 396, 397, 402, 405, 406
- Pie charts, 28, 42, 61, 361, 362, 365, 367, 381
- PieChart, 28, 42, 61, 361, 362, 365, 367, 381
- Plot object classes, 28, 29, 61, 63, 149, 150, 156, 219, 225, 228, 231, 236, 241, 248, 252, 255, 259, 263, 264, 272, 275, 280, 285, 289, 293, 295, 298, 301, 304, 308, 313, 355, 357, 358, 361, 373, 374, 381, 471, 481, 482, 485
- Point3D, 55, 58, 59, 61, 66, 68, 69, 72, 81, 82, 83, 84, 85, 115, 116, 147, 148, 155, 156, 165, 221, 222, 223, 224, 226, 227, 230, 238, 239, 243, 244, 250, 251, 257, 258, 261, 262, 268, 269, 270, 272, 274, 275, 278, 280, 283, 284, 291, 292, 297, 304, 308, 364, 366
- Polysurface, 55, 56, 61
- Polysurface class, 55, 56, 61
- Printing, 54, 61, 413, 414, 415, 416, 417, 418, 419, 420
- Rectangle3D, 55, 56, 58, 61, 110, 114, 401, 403, 404, 414, 418, 419, 420
- Ribbon plots, 45, 61, 219, 225, 227
- Rotate Control, 52, 53, 61
- Rotating Charts, 58, 59, 115, 147, 148, 165, 221, 222, 223, 224, 226, 227, 230, 239, 243, 244, 250, 251, 257, 258, 261, 262, 268, 269, 270, 272, 274, 275, 278, 280, 283, 284, 291, 292, 297, 304, 308, 364, 366
- Scale classes, 17, 18, 61, 105, 107
- Scatter plots, 42, 47, 61, 219, 236, 237, 238, 239, 240, 329
- Shapes, 51, 61, 401, 402, 403, 404, 408, 409, 410
- Simple 3D datasets, 14, 15, 19, 61, 63, 64, 69, 70, 71, 72, 81, 144, 219, 220, 223, 224, 243
- Simple datasets, 14, 15, 16, 19, 58, 61, 63, 64, 65, 66, 67, 68, 69, 72, 73, 77, 81, 120, 121, 122, 123, 125, 142, 219, 220, 223, 224, 225, 228, 229, 230, 232, 237, 344, 345, 361, 362, 364, 366, 434, 442, 464, 467, 485
- Simple plot objects, 12, 28, 42, 44, 45, 46, 47, 48, 51, 52, 53, 61, 81, 149, 219, 225, 228, 231, 237, 241, 250, 253, 257, 260, 264, 266, 282, 286, 296, 361
- SimpleBarPlot, 42, 43, 61, 219, 231, 232, 233, 234, 235, 236, 329, 349
- SimpleDataset, 14, 15, 16, 19, 58, 61, 63, 64, 65, 66, 67, 68, 69, 72, 73, 77, 81, 120, 121, 122, 123, 125, 142, 219, 220, 223, 224, 225, 228, 229, 230, 232, 237, 344, 345, 361, 362, 364, 366, 434, 442, 464, 467, 485

- SimpleDataset3D, 14, 15, 19, 61, 63, 64, 69, 70, 71, 72, 81, 144, 219, 220, 223, 224, 243
- SimpleLineMarkerPlot, 42, 44, 61, 219, 241, 242, 243, 244, 329
- SimpleLinePlot, 42, 46, 61, 219, 220, 221, 222, 223, 224, 228, 327, 329, 349, 435, 442
- SimplePlot, 12, 28, 42, 44, 45, 46, 47, 48, 51, 52, 53, 61, 81, 149, 219, 225, 228, 231, 237, 241, 250, 253, 257, 260, 264, 266, 282, 286, 296, 361
- SimpleRibbonPlot, 45, 61, 219, 225, 227
- SimpleScatterPlot, 42, 47, 61, 219, 236, 237, 238, 239, 240, 329
- Stacked bar plots, 29, 36, 61, 247, 301, 302, 303, 304, 350
- Stacked line plots, 29, 41, 61, 242, 247, 304, 305, 307, 308, 309, 350
- Stacked ribbon plots, 29, 40, 308, 310, 311
- StackedBarPlot, 29, 36, 61, 247, 301, 302, 303, 304, 350
- StackedLinePlot, 29, 41, 61, 242, 247, 304, 305, 307, 308, 309, 350
- StackedRibbonPlot, 29, 40, 308, 310, 311
- Standard legends, 48, 49, 61, 371, 372, 375
- StandardLegend, 48, 49, 61, 371, 372, 375
- String axis labels, 26, 27, 61, 193, 206, 208, 215, 478, 479
- String labels, 50, 61, 208, 381, 392, 395, 396, 397, 398, 399
- StringAxisLabels, 26, 27, 61, 193, 206, 208, 215, 478, 479
- StringLabel, 50, 61, 208, 381, 392, 395, 396, 397, 398, 399
- Symbols, 51, 61, 351, 354
- Text classes, 50, 61, 149, 194, 195, 200, 206, 330, 331, 351, 354, 355, 381, 382, 383, 384, 385, 386, 387, 389, 390, 391, 392, 395, 398, 472
- Tick mark class, 55, 56, 61
- TickMark, 55, 56, 61
- Time auto-scaling, 19, 61
- Time axis, 22, 26, 27, 49, 61, 167, 168, 182, 187, 189, 190, 191, 194, 200, 202, 204, 205, 211, 213, 215, 463, 475, 476, 477
- Time axis labels, 26, 27, 61, 193, 194, 200, 202, 204, 205, 215
- Time coordinates, 17, 18, 58, 61, 105, 108, 115, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 147, 152, 162, 182, 187, 189, 190, 191, 204, 213, 214, 221, 222, 226, 227, 233, 234, 238, 239, 254, 257, 258, 266, 270, 271, 274, 275, 278, 279, 287, 288, 297, 303, 331, 332, 384, 397, 451, 455, 475, 476, 477, 479, 480
- Time labels, 50, 61, 349, 351, 353, 354, 357, 359, 381, 392, 395, 396
- Time scale, 17, 61, 105, 108
- Time/Date group 3D datasets, 14, 16, 19, 61, 63, 64, 99, 100, 101, 103, 104
- Time/Date group datasets, 14, 16, 19, 58, 61, 63, 94, 95, 96, 97, 98, 99, 129, 254, 257, 258, 266, 270, 271, 278, 279, 287, 288, 297, 485
- Time/Date simple 3D datasets, 14, 15, 16, 19, 61, 63, 64, 77, 78, 79, 80, 81, 219
- Time/Date simple datasets, 14, 15, 16, 19, 58, 61, 63, 72, 73, 74, 75, 76, 77, 80, 129, 130, 131, 132, 133, 134, 136, 139, 140, 219, 221, 222, 226, 227, 233, 234, 238, 239, 331, 332, 476, 477, 485, 486
- TimeAxis, 22, 26, 27, 49, 61, 167, 168, 182, 187, 189, 190, 191, 194, 200, 202, 204, 205, 211, 213, 215, 463, 475, 476, 477
- TimeAxisLabels, 26, 27, 61, 193, 194, 200, 202, 204, 205, 215
- TimeCoordinates, 17, 18, 58, 61, 105, 108, 115, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 147, 152, 162, 182, 187, 189, 190, 191, 204, 213, 214, 221, 222, 226, 227, 233, 234, 238, 239, 254, 257, 258, 266, 270, 271, 274, 275, 278, 279, 287, 288, 297, 303, 331, 332, 384, 397, 451, 455, 475, 476, 477, 479, 480
- TimeGroupDataset, 14, 16, 19, 58, 61, 63, 94, 95, 96, 97, 98, 99, 129, 254, 257, 258, 266, 270, 271, 278, 279, 287, 288, 297, 485
- TimeGroupDataset3D, 14, 16, 19, 61, 63, 64, 99, 100, 101, 103, 104
- TimeLabel, 50, 61, 349, 351, 353, 354, 357, 359, 381, 392, 395, 396
- TimeScale, 17, 61, 105, 108
- TimeSimpleDataset, 14, 15, 16, 19, 58, 61, 63, 72, 73, 74, 75, 76, 77, 80, 129, 130, 131, 132, 133, 134, 136, 139, 140, 219, 221, 222, 226, 227, 233, 234, 238, 239, 331, 332, 476, 477, 485, 486
- TimeSimpleDataset3D, 14, 15, 16, 19, 61, 63, 64, 77, 78, 79, 80, 81, 219
- ToolTips, 52, 53, 61, 349, 350, 351, 352, 353, 354, 355, 356, 472
- User coordinates, 17, 18, 61, 105, 107
- UserControl, 12, 13, 14, 61, 143, 151, 155, 425, 431, 447, 449, 453, 472, 479, 482
- UserCoordinates, 17, 18, 61, 105, 107
- Visual Basic, 6, 9, 65, 66, 68, 69, 70, 71, 72, 73, 74, 76, 77, 78, 79, 80, 81, 82, 83, 85, 86, 88, 89, 90, 91, 92, 93, 94, 95, 97, 98, 99, 100, 103, 110, 111, 112, 114, 115, 116, 118, 119, 120, 121, 122, 124, 125, 128, 129, 130, 132, 133, 134, 135, 136, 138, 139, 140, 141, 142, 143, 144, 145, 150, 151, 152, 153, 155, 156, 160, 162, 163, 168, 171, 172, 174, 175, 177, 178, 180, 181, 187, 188, 189, 191, 196, 197, 198, 199, 202, 203, 204, 205, 206, 207, 211, 212, 214, 220, 221, 223, 225, 227, 228, 232, 234, 236, 237, 239, 240, 242, 243, 244, 249, 251, 253, 254, 256, 258, 260, 261, 263, 265, 266, 268, 271, 273, 276, 279, 281, 283, 286, 288, 289, 291, 293, 294, 296, 297, 299, 300, 302, 303, 305, 307, 309, 311, 315, 318, 321, 323, 325, 327, 329, 332, 333, 334, 338, 340, 341, 344, 347, 350, 352, 353, 354, 358, 361, 365, 372, 373, 376, 377, 379, 383, 387, 388, 389, 390, 391, 392, 395, 396, 398, 402, 404, 405, 407, 408, 409, 413, 414, 417, 421, 423, 425, 431, 440, 453, 472, 475, 477, 478, 479, 480, 481, 483, 485
- Visual C#, 9, 427, 432, 440, 448, 449
- Wall3D, 21, 22, 61, 149, 157, 163, 164, 165, 166
- Working coordinates, 17, 18, 61, 105, 107, 109
- WorkingCoordinates, 17, 18, 61, 105, 107, 109
- World coordinates, 17, 18, 61, 105, 107

WorldCoordinates, 17, 18, 61, 105, 107
Zoom, 346

Zooming, 52, 53, 61, 337, 338, 339, 340, 341, 342,
343, 344, 346, 470