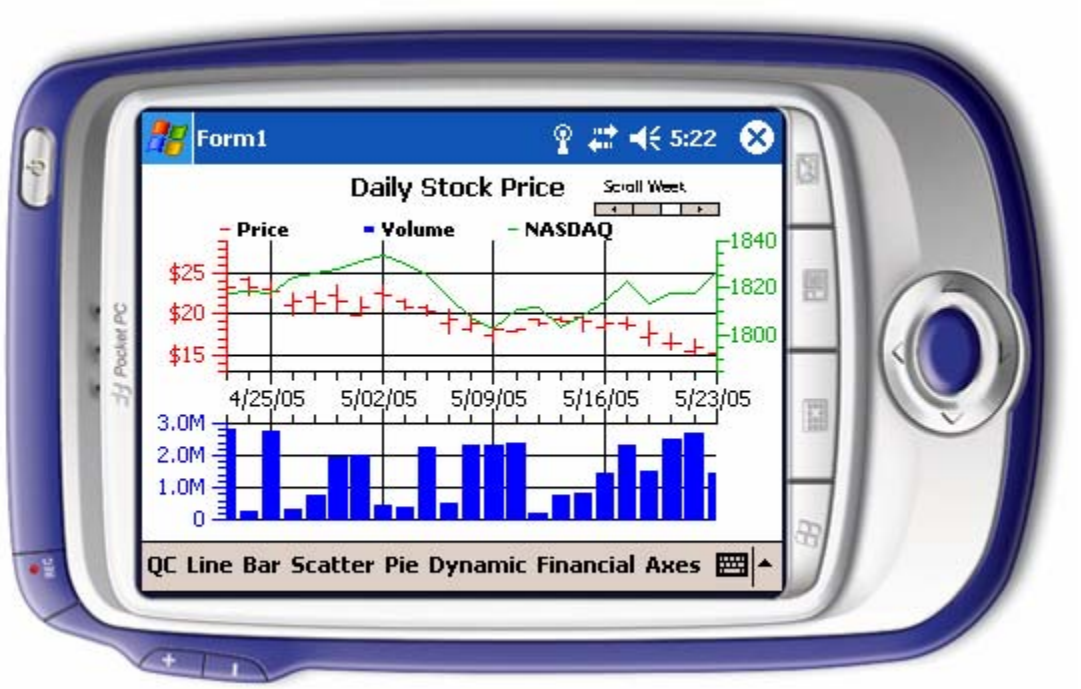
QCChart2D CF Charting Library



for the .Net Compact Framework

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 6/1/2009 Rev. 2.0

Quinn-Curtis, Inc. Tools for *.Net Compact Framework* 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 all Quinn-Curtis, Inc. ***.Net Compact Framework*** software (on any media) and related documentation (on any media). 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. polices 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 Hearthstone 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.

1. Introduction – Charting Tools for the .Net Compact Framework Charting Library

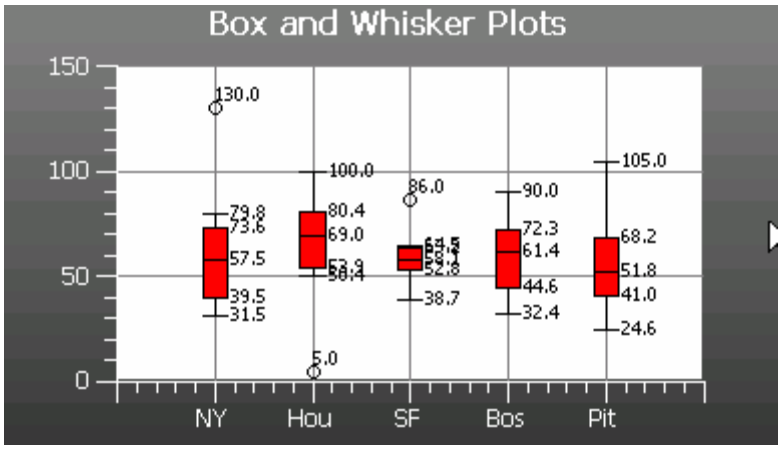
New Features found in the 2.0 version of QCChart2D

Many new features have been added to Revision 2.0 of QCChart2D for .Net CF. Existing classes have been extended, and new classes added. Features new to Revision 2.0 include:

- Five new plot types for the Cartesian, time and elapsed Time coordinate systems: **BoxWiskerPlot**, **FloatingStackedBarPlot**, **RingChart**, **SimpleVersaPlot**, and **GroupVersaPlot**
- Antenna Charts - includes coordinate system (**AntennaCoordinates**), axes (**AntennaAxes**), axes labels (**AntennaAxesLabels**), grid (**AntennaGrid**) and plot objects (**AntennaLinePlot**, **AntennaScatterPlot**, **AntennaLineMarkerPlot** and **AntennaAnnotation**).
- Elapsed time scaling to compliment the time/date scaling. This includes new dataset types (**ElapsedTimeSimpleDataset**, **ElapsedTimeGroupDataset**), a new coordinate system class (**ElapsedTimeCoordinates**), a new axis class (**ElapsedTimeAxis**) and a new axis labels class (**ElapsedTimeAxisLabels**).
- Vertical axis scaling for time/date and elapsed time
- A **DatasetViewer** class for the grid-like display of dataset information in a table.
- The **MagniView** class represents a new way to zoom data
- A **CoordinateMove** class is used to pan the coordinate system, left, right, up, down.
- New **ChartZoom** feature add integrated zoom stack processing, and fixed aspect ratio zooming.

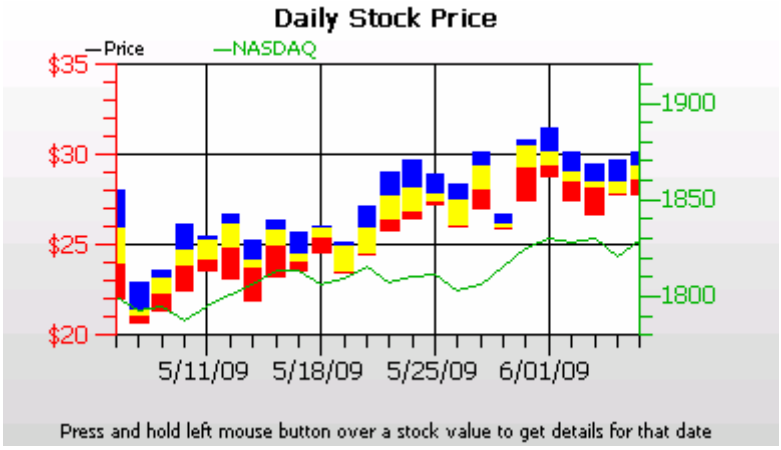
New Plot Types

Five new general plot types have been added to the software for use with **CartesianCoordinates**, **TimeCoordinates** and **ElapsedTimeCoordinates**: box and whisker plots (**BoxWhiskerPlot**), floating stacked bar plots (**FloatingStackedBarPlot**), ring charts, simple versa plots (**SimpleVersaPlot**), group versa plots (**GroupVersaPlot**).



A Box and Whisker plot

The **BoxWhiskerPlot** class graphically depicts groups of numerical data through their five-number summaries (the smallest observation, lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation).



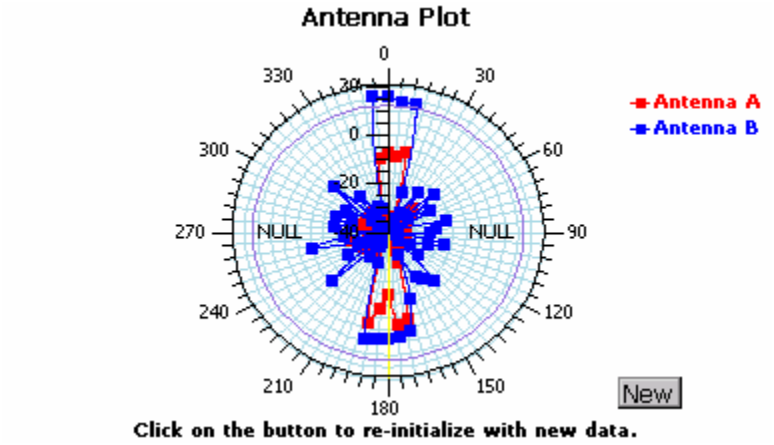
A Floating Stacked Bar plot

In the **FloatingStackedBarPlot**, the bars 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.

A Ring chart

The ring chart is a similar to a pie chart. It uses segmented rings instead of pie shaped wedges.

The **SimpleVersaPlot** plot type can look like any of the simple plot types while the **GroupVersaPlot** type can look like any of the group plot types.



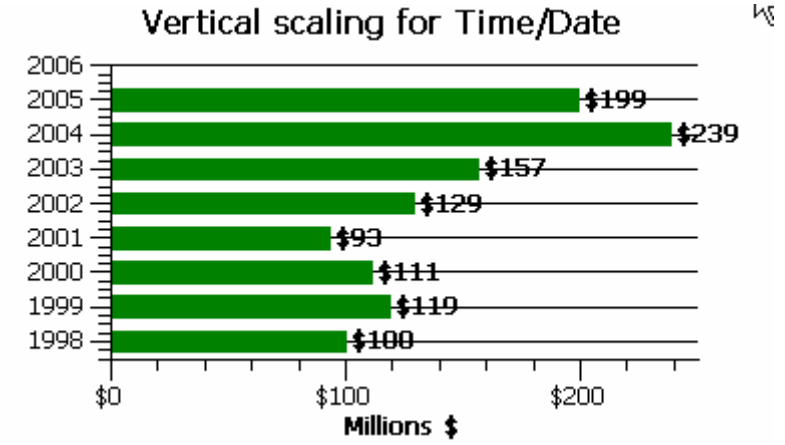
Antenna chart

An Antenna chart (named Antenna because it is commonly used by engineers plotting the performance characteristics of an antenna design) is a circular chart type similar to a polar chart. Data is specified using a pair of values, (radius, angle), where the angle is in degrees. 12:00 is zero degrees and the angle increases clockwise. This contrasts to a polar chart, where 3:00 is zero radians and the angle increases counter-clockwise. Also, antenna radar chart allows plus/minus scaling of the radius, for example: the radius can be scaled from -40 to 20, as in the picture above. The Antenna chart uses a unique

coordinate system (**AntennaCoordinates**), axes (**AntennaAxes**), axes labels (**AntennaAxesLabels**), grid (**AntennaGrid**) and plot objects (**AntennaLinePlot**, **AntennaScatterPlot**, **AntennaLineMarkerPlot** and **AntennaAnnotation**).

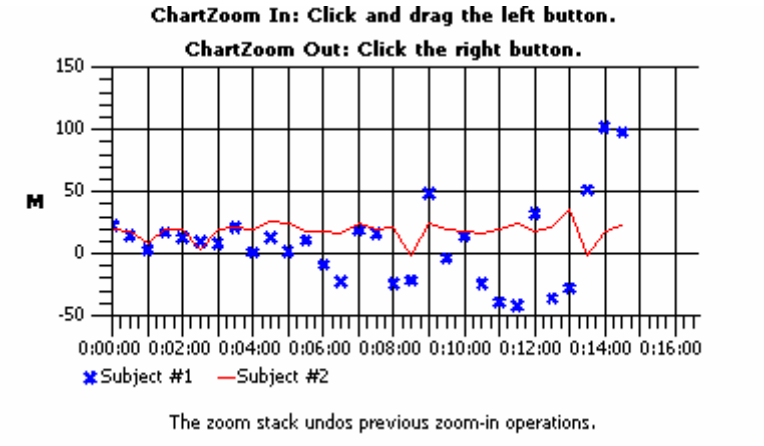
New Coordinate System Features

A TimeCoordinates coordinate system can now have a vertical time/date scale
The **TimeCoordinates** class now supports time/date scaling for both the x- and y-axis of a chart.



A bar chart with vertical time/date axis

Elapsed Time support for linear and logarithmic coordinate systems
The **TimeCoordinates** class proved less than optimal for the display of simple elapsed time scales. The software now supports elapsed time scales with the addition of **ElapsedTimeCoordinates**, **ElapsedTimeScale**, **ElapsedTimeAutoScale**, **ElapsedTimeAxis**, **ElapasedTimeLabel**, and **ElapsedTimeAxisLabels** classes. For example, you can now have a scale with a range of 00:00:00 to 12:00:00, without an explicit calendar date associated with it.



Elapsed Time support

New User Interface Features

Integrated MultiMouseListener

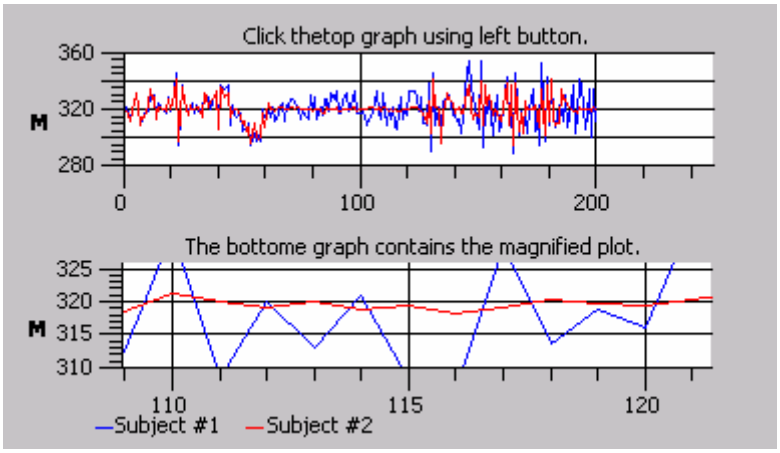
The **ChartView** class supports multiple mouse listeners through the integration of our **MultiMouseListener** class.

The ChartZoom class

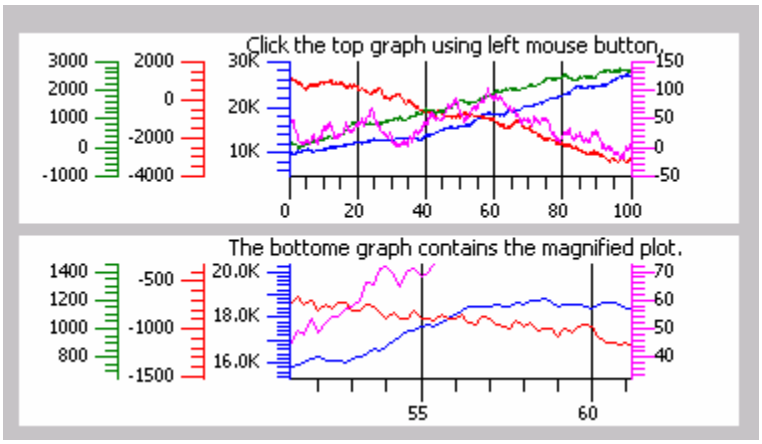
Zoom stack processing has been added to the **ChartZoom** class. You no longer have to subclass the **ChartZoom** class to add zoom stack processing to your application. Also, a new mode forces the aspect ratio of zoomed graphs to remain fixed, traking either the x or y-dimension of the zoom rectangle.

The MagniView class

A new **MagniView** class is similar to zooming. The mouse controls a "magnifying" rectangle as it passes over an existing chart. The area of the chart within the bounds of the magnifying rectangle is "magnified" and continuously redrawn in a separate window.



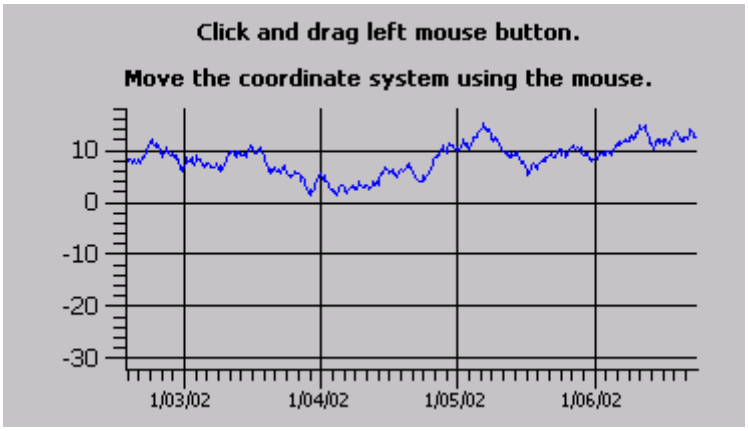
*In use the **MagniView** class is similar to passing a magnifying glass over your data*



*The **MagniView** class works with an unlimited number of coordinate systems and axes.*

The MoveCoordinates class

A new **MoveCoordinates** class is analogous to panning of the graph. Instead of using a scroll bar to set the starting point of the scale in a graph, the mouse now "moves" the coordinate system, without changing the range of the x- and y-scales. The x- and y-axes are continuously redrawn as the mouse is dragged, reinforcing the look and feel of the move operation. This is similar to the way you use a mouse to "move" a map under Map Quest or Google.



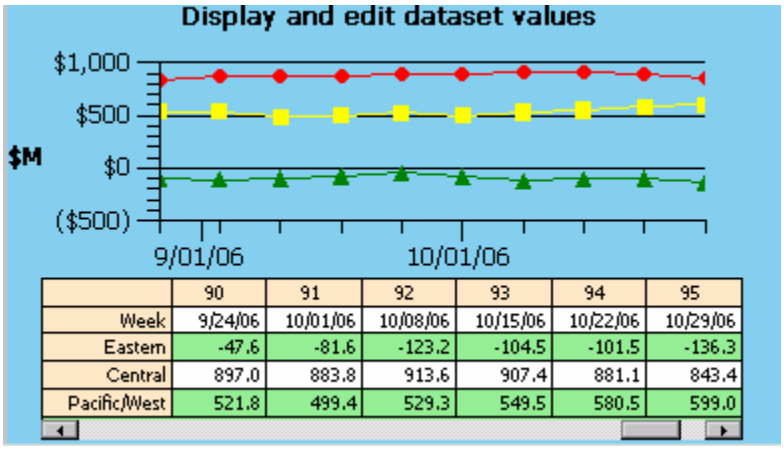
*The **MoveCoordinates** class is an alternative to using scroll bars to pan the graph left, right, up, down.*

Additional Chart Objects

Integrated data grids for viewing chart data.

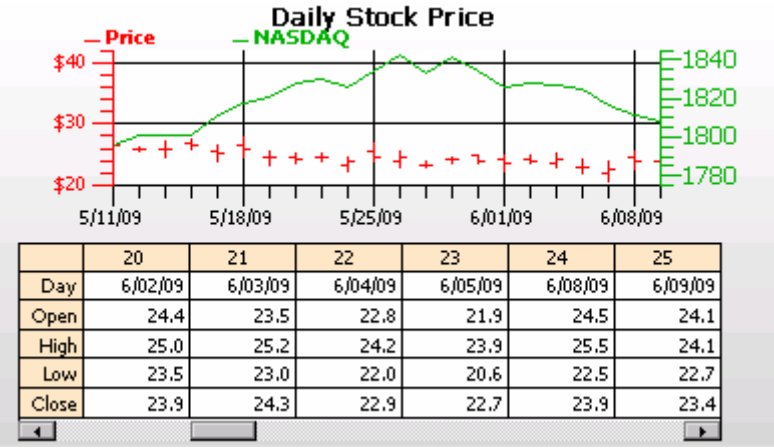
The new **DatasetViewer** class displays chart datasets using a simple grid, or table format. Horizontal and vertical scrolling options are supported. Numeric and time/date based formats are also supported. Row and column headers can be customized. Individual grid

cells can be edited and the associated chart immediately updated using the synchronize feature.



	90	91	92	93	94	95
Week	9/24/06	10/01/06	10/08/06	10/15/06	10/22/06	10/29/06
Eastern	-47.6	-81.6	-123.2	-104.5	-101.5	-136.3
Central	897.0	883.8	913.6	907.4	881.1	843.4
Pacific/West	521.8	499.4	529.3	549.5	580.5	599.0

DatasetViewer *displaying the contents of a* ***TimeSimpleDataset****.*



	20	21	22	23	24	25
Day	6/02/09	6/03/09	6/04/09	6/05/09	6/08/09	6/09/09
Open	24.4	23.5	22.8	21.9	24.5	24.1
High	25.0	25.2	24.2	23.9	25.5	24.1
Low	23.5	23.0	22.0	20.6	22.5	22.7
Close	23.9	24.3	22.9	22.7	23.9	23.4

DatasetViewer *display the contents of a* ***TimeGroupDataset****.*

Visual Studio 2005 Projects

All of the example program projects have been converted to the Visual Studio 2005 project format. The VS 2005 project format is the oldest project format we expect to support in years moving forward. While it may be possible to use the QCChart2D software with VS 2003, or VS 2002, it is not something we support any more. You should assume that as we continue to enhance the software, we will use features not supported under VS 2003/2002. We can still sell you a Revision 1.7 version of the

software, with all of the original VS 2003/2002 projects; however it will not include the features described above.

One difference betwee of VS 2005 and VS 2003/2002 is the VS 2005 creation of *partial* classes when using the **Add User Control** wizard. The VS 2005 **Add User Control** wizard now creates two classes, UserControlName.cs and UserControlName.Designer.cs (or UserControlName.vb and UserControlName.Designer.vb), by default. The Designer specific code is now placed in the UserControlName.Designer.vb file. In VS 2003/2002, where a separate UserControlName.Designer.cs file is NOT created, the Designer code was placed in the main UserControlName.cs file, most of which was hidden using the

```
#region Windows Form Designer generated code
```

compiler directive. Many of example programs still use this older style, with the single UserControlName file. The single file structure is forward compatible with the VS 2005 compiler. All of the example programs demonstrating new features in the software use the split UserControlName.cs/UserControlName.Designer.cs file structure. This split structure is NOT backward compaticle with VS 2003/2002.

Differences between this and the previous (1.6) version.

Starting with Visual Studio 2005, the .Net Compact Framework 2.0 supports the **UserControl** class. In the 1.6 version of the software, the **ChartView** control was derived from the .Net CF **Control** class and this has been changed to **UserControl** in order to make the .Net CF version of QCChart2D source compatible with the .Net version. Since the **ChartView** control is now derived from **UserControl**, and because of improvements made in the .Net CF design mode, you can now drop and drag the **ChartView** control onto a .Net CF form.

Also, starting with .Net CF 2.0, line drawing supports line thicknesses greater than one, and line styles (solid, and dashed). We have enabled this feature in the **ChartAttribute** class, so your .Net CF charts can now have thick lines, and lines styles (solid and dashed).

In the original 1.6 version of the software, the QCChart2DNetCF library defined a **DashStyle** enumeration to make up for the one included with .Net 1.0, but not .Net CF 1.0. Since .Net CF 2.0 now includes a **DashStyle** enumeration we have removed the one in the QCChart2DNetCF library, and use the System.Drawing.Drawing2D enumeration of **DashStyle** exclusively. So, any modules that reference **DashStyle** must include a reference to System.Drawing.Drawing2D in the *using* (C#) or *Imports* (VB) section of the program.

We have eliminated the QCLicense license file from the software and with it the need to purchase additional Redistributable Licenses. Once you purchase the software, you the developer, can create application programs that use this software and redistribute the programs and our libraries royalty free. As a development tool, i.e. using this software in

conjunction with a compiler, the software is still governed by a single user license and cannot be shared by multiple individuals unless additional copies, or a site license, have been purchased.

We have discontinued support for programming .Net CF applications using Visual Studio 2003 and no projects for VS 2003 have been included with the software. The current version of the software has been tested to be compatible with VS 2005 and VS 2008.

Tutorials

Tutorials that describe how to get started with the **QCChart2D CF** for the **.Net Compact Framework** charting software are found in Chapter 23 (*Using QCChart2D CF for the .Net Compact Framework to Create Windows Applications*).

.Net Compact Framework Background

The goal of the Microsoft **.Net Compact Framework** is, according to Microsoft:

"The .Net Framework and the .Net Compact Framework provide a consistent programming model across the full range of Windows platforms and expose that programming model through a single unified tool set, Visual Studio .Net. Together, Visual Studio . and the .Net Compact Framework enable millions of desktop Visual Basic developers and the rapidly growing market of C# developers to begin building smart mobile applications. Features of the .Net Compact Framework and Visual Studio .Net include support for XML and Web services; the ability to integrate components written in multiple programming languages; and developer productivity features such as integrated rich device emulator support, a visual drag-and-drop forms designer, a comprehensive set of user interface controls, remote debugging support, and simplified application deployment. "

Limitation of the .Net API Compact Framework API

Microsoft **.Net Compact Framework** includes a basic 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 subset of the standard .Net graphics API, supporting far fewer classes than are found in that API, and far fewer methods and properties in the classes that are supported. A few of the limitations of the **.Net Compact Framework** API, compared to the regular .Net API are:

- Brushes only support simple color, i.e. no gradients, or textures
- Only simple RGB colors are supported with no alpha blending (no transparency)
- No generalized geometry support for arbitrary shapes because **.Net Compact Framework** lacks a Matrix class and a GraphicsPath class

- No 2D coordinate transformation classes, for rotating text and arbitrary geometric shapes
- Text cannot be rotated, not even 90 degrees for vertical text
- No printer and image output support
- Images imported into a program cannot be rotated
- In general, none of the advanced features found in the System.Drawing.Drawing2D library are available to the **.Net Compact Framework** programmer.

The **QCChart2D CF** for the **.Net Compact Framework** software derives from our original **QCChart2D** for .Net software. The original software makes extensive use of the advanced graphics features found in the workstation version of .Net. In order to port the **QCChart2D** software to the **.Net Compact Framework** compromises had to be made because of the limitations listed above. In some cases, work-arounds were devised to overcome these limitations. When compared to the original **QCChart2D** for .Net software, the **QCChart2D CF** for the **.Net Compact Framework** has the following limitations:

- While the Brush class only supports a simple color, i.e. no gradients, or textures, we were able to implement simple linear gradients for our ChartBackground class
- Only simple RGB colors are supported with no alpha blending (no transparency)
- Because much of the original software was written using the GraphicsPath class, we created our own simple GraphicsPath class to maintain source compatibility
- A simple Matrix class, along with related 2D coordinate transformation routines, were added so that we could rotate, scale and translate geometric shapes and symbols defined using GraphicsPath objects.
- Image objects incorporated in a chart can be scaled and translated, but not rotated.
- Text still cannot be rotated, not even 90 degrees for y-axis titles
- *We implemented XOR drawing so that we could properly draw zoom rectangles and data cursor objects
- No printer and image output support

*The XOR drawing is optional. We implement XOR drawing by calling a **.Net CF PInvoke** function that permits calling drawing routines in the underlying oprerating system. Strickly speaking, this violates the .Net managed memory paradigm, though it never really mattered in Revisions 1.0, 1.1 and 2.0 of .Net CF. For unknown reasons, starting with the .Net CF 3.5, found in Visual Studio 2008, it does matter. The VS 2008 Toolbox will not host a UserControl derived object that contains PInvoke calls. Therefore, we include two versions of the QCChart2DNetCF DLL, one that uses PInvoke and supports XOR drawing, and one that does not use PInvoke. They are named QCChart2DNetCF_XOR.DLL.(uses PInvoke and support XOR drawing mode) and QCChart2DNetCF_NOXOR.DLL .(does not use PInvoke and does not support XOR drawing mode). The DLL you wish to use must copied and renamed to QCChart2DNetCF.DLL and placed in the Quinn-Curtis\DotNet\lib folder. The default QCChart2DNetCF.DLL file is a copy of QCChart2DNetCF_NOXOR.DLL.

QCChart2D CF for .Net Compact Framework Dependencies

The **QCChart2D CF** for the **.Net Compact Framework** class library is self-contained. It uses only standard classes that ship with the Microsoft **.Net Compact Framework** 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 basic graphics functionality.

System.Drawing.Drawing2D Class

The System.Drawing.Drawing2D namespace contains the DashStyle enumerication that is used by many of the programs. The VS 2003 version of.Net CF SDK did not support dash styles for line drawing and the System.Drawing.Drawing2D ws not used.

System.Drawing.Imaging Namespace

The **System.Drawing.Imaging** namespace implements basic 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 RGB (Red, Green, Blue) components.

System.Drawing.Font Class

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

System.IO Namespace

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

System.Collections Namespace

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

.

Installation Directory Structure

The **QCChart2D CF** for the **.Net Compact Framework** 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

QCChart2DNetCFCompiledHelpFile.chm – the QCChart2D CF compiled help file

QCChart2DNetCFManual.pdf – PDF file of the QCChart2D CF Programming Manual

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

QCChart2D CF\ - Language specific code directory

CSharp\ - C# specific directory

CF Examples\ - C# examples directory

BankAmerica

BetaPictoris

BigChartDemo\ - Principle c# example

Boston Climate

BushGuardRecords

EuroDollar

JapanWorkforce

LandOfTheFry

MedicareDrugCosts

NasaSpending

NewDemosRev2 – New examples for Revision 2.0 features.

RealGDPGrowth

TechnicalAnalysis

UserChartExample1\ - Simple example

UserChartExample2\ - Simple example

UserChartExample3\ - Simple example

ZoomExamples\ - Examples for zooming, MagniView and Coordinate Moving

VB\ - VB specific code

CF Examples\ - VB examples

BankAmerica

BetaPictoris

BigChartDemo\ - Principle VB example

BostonClimate

BushGuardRecords

EuroDollar

JapanWorkforce

LandOfTheFry

MedicareDrugCosts

NasaSpending

NewDemosRev2 – New examples for Revision 2.0 features.

RealGDPGrowth

TechnicalAnalysis

UserChartExample1\ - Simple example

UserChartExample2\ - Simple example

UserChartExample3\ - Simple example

ZoomExamples\ - Examples for zooming, MagniView and Coordinate Moving

There are two versions of the **QCChart2D CF** for the **.Net Compact Framework** class library: the 30-day trial versions and the developers, summarized below:

30-Day Trial Version

The trial version of **QCChart2D CF** for the **.Net Compact Framework** is downloaded as a zip file named Trial_QCChart2DCFR18x.zip. The 30-day trial version stops working 30 days after the initial download. The trial version may include a version message in the upper left corner of the graph window that cannot be removed.

Developer Version

The subscription version of **QCChart2D CF** for the **.Net Compact Framework** is downloaded as a zip file, name something similar to NETCFCHTDEV1R1x0x353x1.zip. The developer version does not time out and you can use it to create application programs that you can distribute royalty free. You can download free updates for a period of 2-years. When you placed your order, you were e-mailed download link(s) that will download the software. Those download links will remain active for at least 2 years and should be used to download current versions of the software. After 2 years you may have to purchase an upgrade to continue to download current versions of the software

Chapter Summary

The remaining chapters of this book discuss the **QCChart2D CF** for the **.Net Compact Framework** interactive charting package.

Chapter 2 presents the overall class architecture of the **QCChart2D CF for the .Net Compact Framework** 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, time and polar 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, scatter plots., line-marker plots and simple versa 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, candlestick plots, box and whisker plots, floating stacked bar plots, and group versa 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 and ring charts.

Chapter 18 describes classes for the display of data in a polar, and antenna, chart format.

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 the DatasetViewer class is used to display simple and group datasets in a table format.

Chapter 22 describes how to use a generalized shape class for the display of arbitrary lines, shapes, images and arrows.

Chapter 23 is a tutorial that describes how to use **QCChart2D CF** to create Windows applications using Visual Studio .Net, Visual C# and Visual Basic.

Chapter 24 is a collection of Frequently Asked Questions about **QCChart2D CF** for **.Net Compact Framework**.

2. Class Architecture of the QCChart2D CF for the .Net Compact Framework Class Library

Major Design Considerations

This chapter presents an overview of the **QCChart2D CF** for the **.Net Compact Framework** class architecture. It discusses the major design considerations of the architecture:

- It is based on 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 chapter also summarizes the classes in the **QCChart2D CF** for the **.Net Compact Framework** library.

There are five primary features of the overall architecture of the **QCChart2D CF** for the **.Net Compact Framework** classes. These features address major shortcomings in existing charting software for use with both .Net and other computer languages.

- First, **QCChart2D CF** for the **.Net Compact Framework** 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 a.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 **QCChart2D CF** for the **.Net Compact Framework** 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 do 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. **QCChart2D CF** for the **.Net Compact Framework** 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 **QCChart2D CF** for the **.Net Compact Framework** architecture includes classes that implement the .Net CF **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.

QCChart2D CF for .Net Compact Framework 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 and group data types. There are also data classes that handle **System.DateTime** date/time data and contour 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, 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 2D 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.
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 CF **System.Windows.Forms.UserControl** class**,** where the **Control** class is the base class for the .Net CF 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 **Control** class, it acts as a container for other .Net CF components too.

Data Classes

ChartDataset
SimpleDataset
TimeSimpleDataset
ElapsedTimeSimpleDataset
ContourDataset
GroupDataset
TimeGroupDataset
ElapsedTimeGroupDataset

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 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.

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.
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.

ElapsedTimeSimpleDataset A subclass of **SimpleDataset**, it is initialized with **TimeSpan** objects, or milliseconds, in place of the x- or y-values.

ContourDataset A subclass of **SimpleDataset**, it adds a third dimension (z-values) to the x- and y- values of the simple dataset.

GroupDataset Represents group data, where every x-value can have one or more y-values.

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.

ElapsedTimeGroupDataset A subclass of **GroupDataset**, it uses **TimeSpan** objects, or milliseconds, as the x-values, and floating point numbers as the y-values.

Scale Classes

ChartScale
- **LinearScale**
- **LogScale**
- **TimeScale**
- **ElapsedTimeScale**

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- and y-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.

ElapsedTimeScale A concrete implementation of the **ChartScale** class. converts an elapsed time coordinate system into the working coordinate system.

Coordinate Transform Classes

UserCoordinates
WorldCoordinates
WorkingCoordinates
PhysicalCoordinates
CartesianCoordinates
ElapsedTimeCoordinates
PolarCoordinates
AntennaCoordinates
TimeCoordinates

The coordinate transform classes maintain a 2D coordinate system. Many 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.

UserCoordinates This class manages the interface to the **System.Drawing** classes and contains routines for drawing lines, rectangles and text using .Net CF device coordinates.

WorldCoordinates This class derives from the **UserCoordinates** class and maps a device independent world coordinate system on top of the .Net CF device coordinate system.

WorkingCoordinates

This class derives from the **WorldCoordinates** class and extends the physical coordinate system of the *plot area* (the area typically bounded by the charts axes) to include the complete *graph area* (the area of the chart outside of the *plot area*).

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 into a working coordinate system. Different scale objects

(**ChartScale** derived) are installed for converting physical x- and y-coordinate values into working coordinate values.

CartesianCoordinates

This class is a concrete implementation of the **PhysicalCoordinates** class and implements a coordinate system used to plot linear, logarithmic and semi-logarithmic graphs.

TimeCoordinates This class is a concrete implementation of the **PhysicalCoordinates** class and implements a coordinate system used to plot **GregorianCalenar** time-based data.

ElapsedTimeCoordinates This class is a subclass of the **CartesianCoordinates** class and implements a coordinate system used to plot elapsed time data.

PolarCoordinates This class is a subclass of the **CartesianCoordinates** class and implements a coordinate system used to plot polar coordinate data.

AntennaCoordinates This class is a subclass of the **CartesianCoordinates** class and implements a coordinate system used to plot antenna coordinate data. The antenna coordinate system differs from the more common polar coordinate system in that the radius can have plus/minus values, the angular values are in degrees, and the angular values increase in the clockwise direction.

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, 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**
- **ElapsedTimeAutoScale**

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 class is a concrete implementation of the **AutoScale** class. It calculates scaling values based on the numeric values in **SimpleDataset** and **GroupDataset** objects. Linear scales and axes use it for auto-scale calculations.
LogAutoScale	This class is a concrete implementation of the **AutoScale** class. It calculates scaling values based on the numeric values in **SimpleDataset** and **GroupDataset** objects. Logarithmic scales and axes use it for auto-scale calculations.
TimeAutoScale	This class is a concrete implementation of the **AutoScale** class. It calculates scaling values based on the **ChartCalendar** values in **TimeSimpleDataset** and **TimeGroupDataset** objects. Date/time scales and axes use it for auto-scale calculations.
ElapsedTimeAutoScale	This class is a concrete implementation of the **AutoScale** class. It calculates scaling values based on the numeric values in **ElapsedTimeSimpleDataset** and **ElapsedTimeGroupDataset** objects. The elapsed time classes 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.

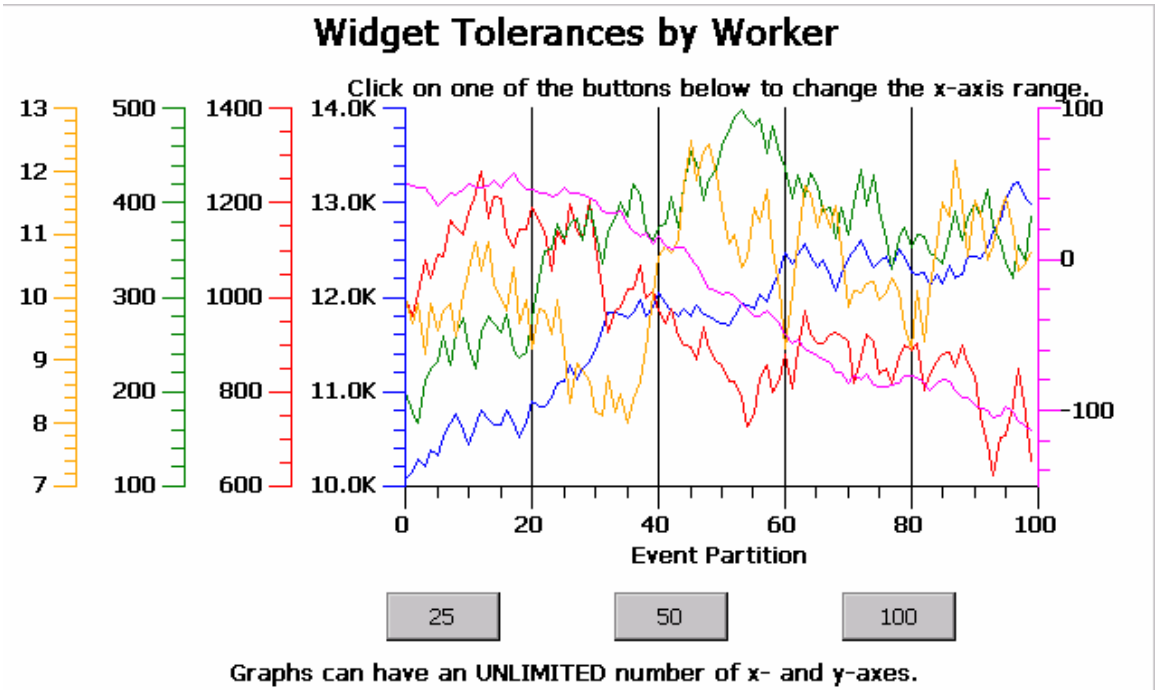
GraphObj	This class is the abstract base class for all drawable graph objects. It contains information common to all chart objects. This class includes references to instances of the **ChartAttribute** and **PhysicalCoordinates** classes. The **ChartAttribute** class contains basic color and line style 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, or the plot area of the chart, using a solid color, or a color or a gradient.

Axis Classes

Axis
- **LinearAxis**
  - **PolarAxes**
  - **AntennaAxes**
  - **ElapsedTimeAxis**
- **LogAxis**
- **TimeAxis**

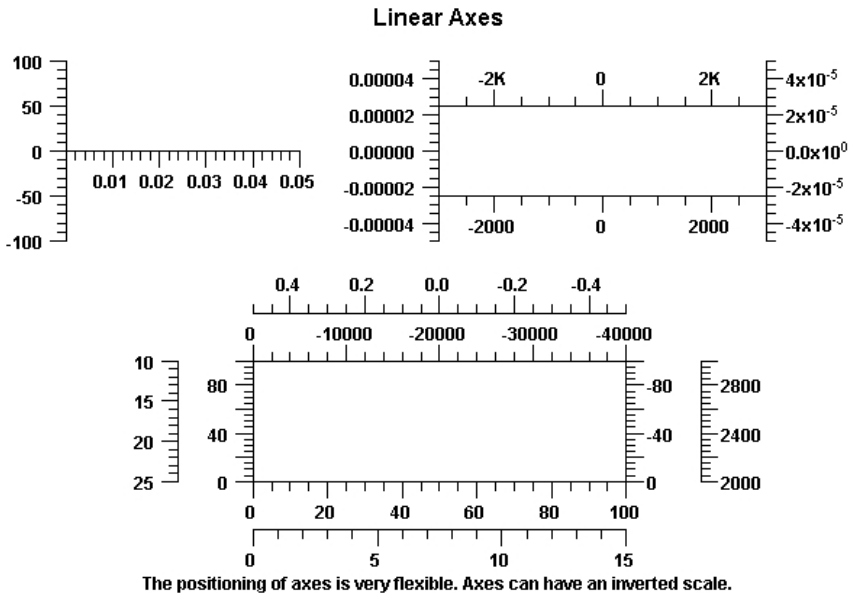
Creating a **PhysicalCoordinates** coordinate system does not automatically create a pair of x- and y-axes. Axes are separate charting objects drawn with respect to a specific **PhysicalCoordinates** object. The coordinate system and the axes do not need to have the same limits. In general, the limits 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.

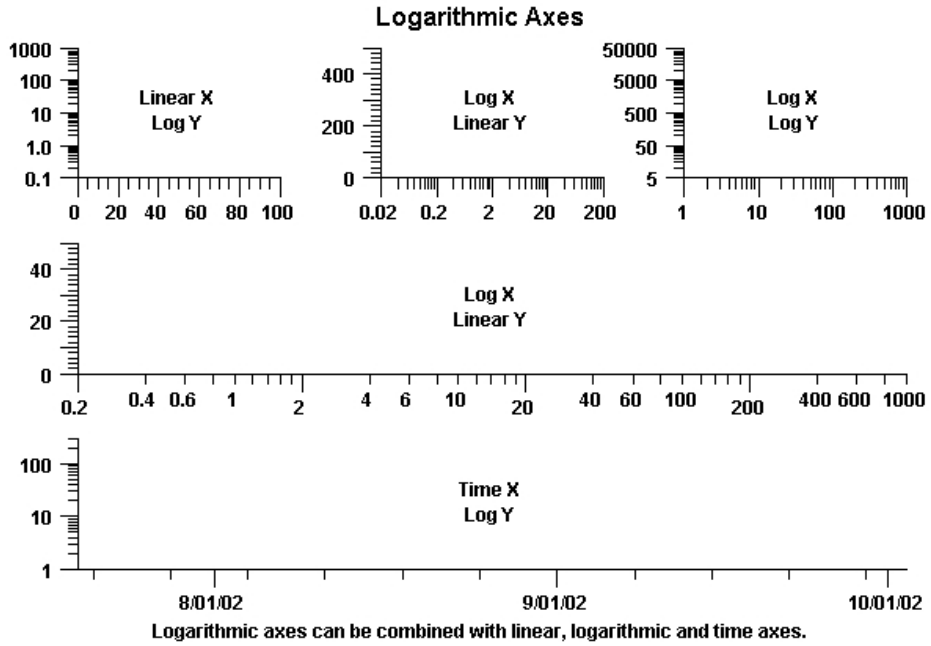


Axis

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

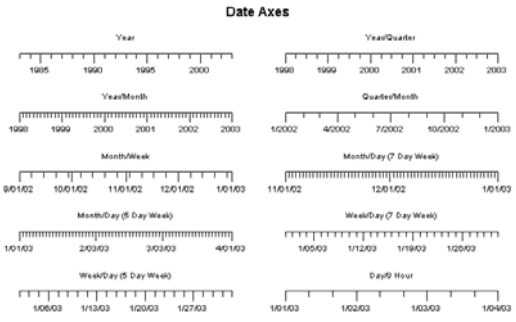


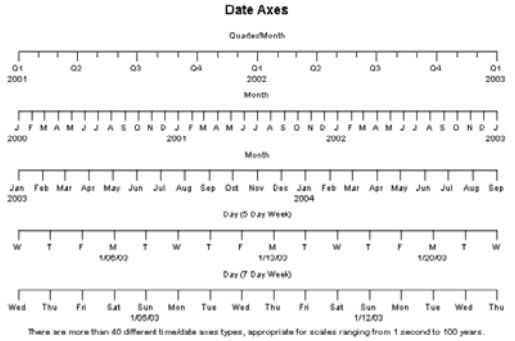
LinearAxis This class implements a linear axis with major and minor tick marks placed at equally spaced intervals.

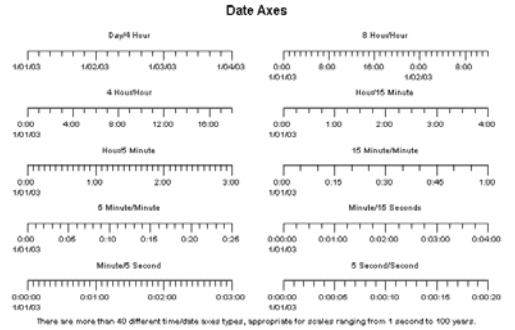


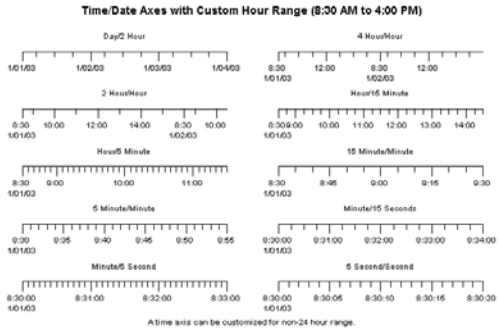
LogAxis

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.



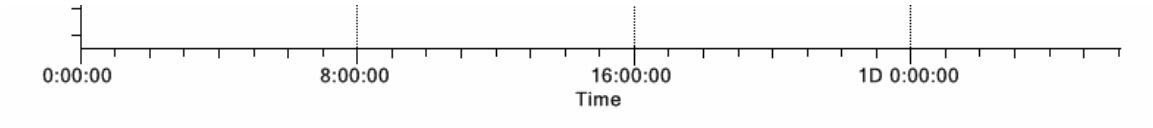






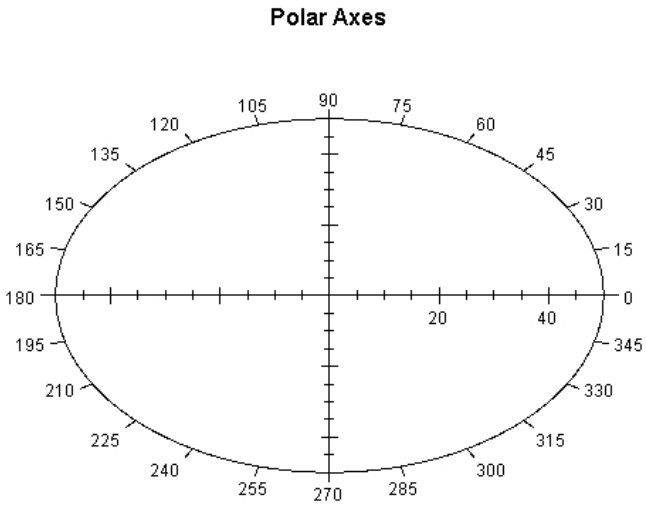
TimeAxis

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 .Net CF **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.



ElapsedTimeAxis

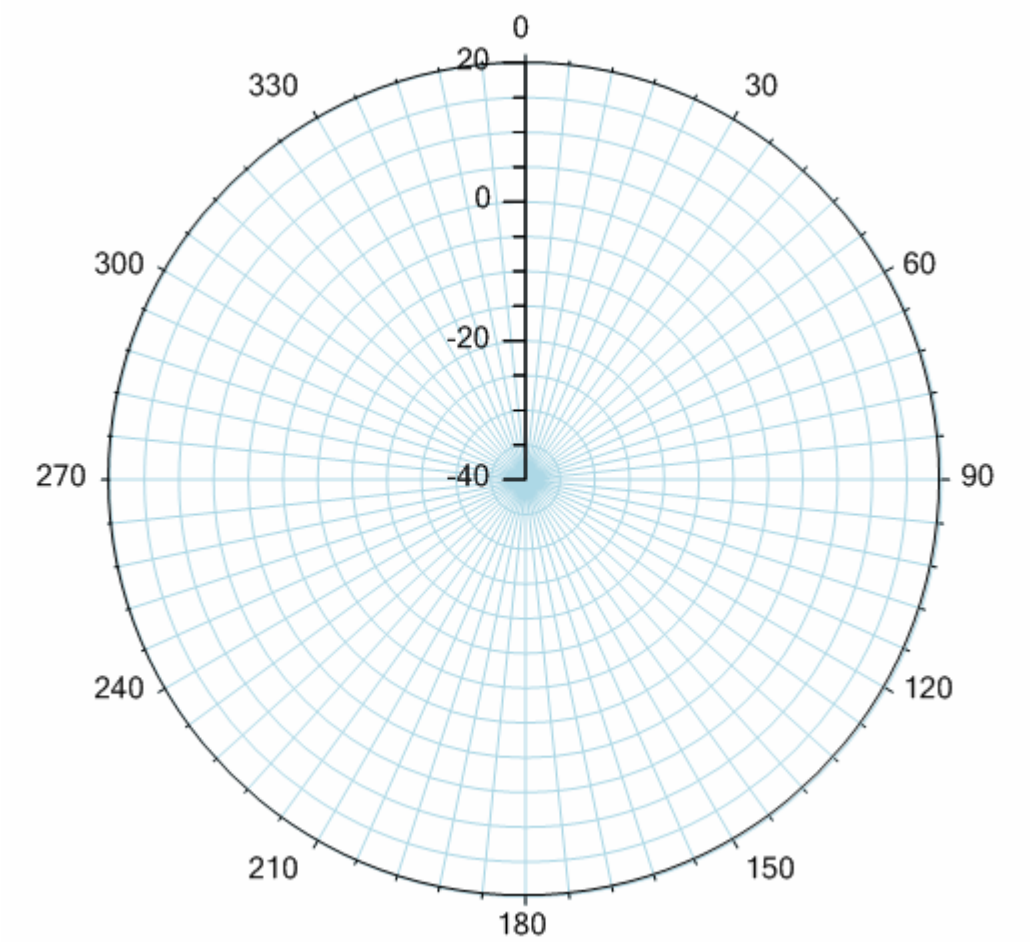
The elapsed time axis is very similar to the linear axis and is subclassed from that class. The main difference is the major and minor tick mark spacing calculated by the CalcAutoAxis method takes into account the base 60 of seconds per minute and minutes per hour, and the base 24 of hours per day. It is a continuous linear scale.



A polar axis consists of the x and y axis for magnitude, and the outer circle for the angle.

PolarAxes

This class has three separate axes: two linear and one circular. The two linear axes, scaled for +- the magnitude of the polar scale, form a cross with the center of both axes at the origin (0, 0. The third axis is a circle centered on the origin with a radius equal to the magnitude of the polar scale. This circular axis represents 360 degrees (or 2 Pi radians) of the polar scale and the tick marks that circle this axis are spaced at equal degree intervals.



AntennaAxes

This class has two axes: one linear y-axis and one circular axis. The linear axis is scaled for the desired range of radius values. This can extend from minus values to plus values. The second axis is a circle centered on the origin with a radius equal to the range of the radius scale. This circular axis represents 360 degrees of the antenna scale and the tick marks that circle this axis are spaced at equal degree intervals.

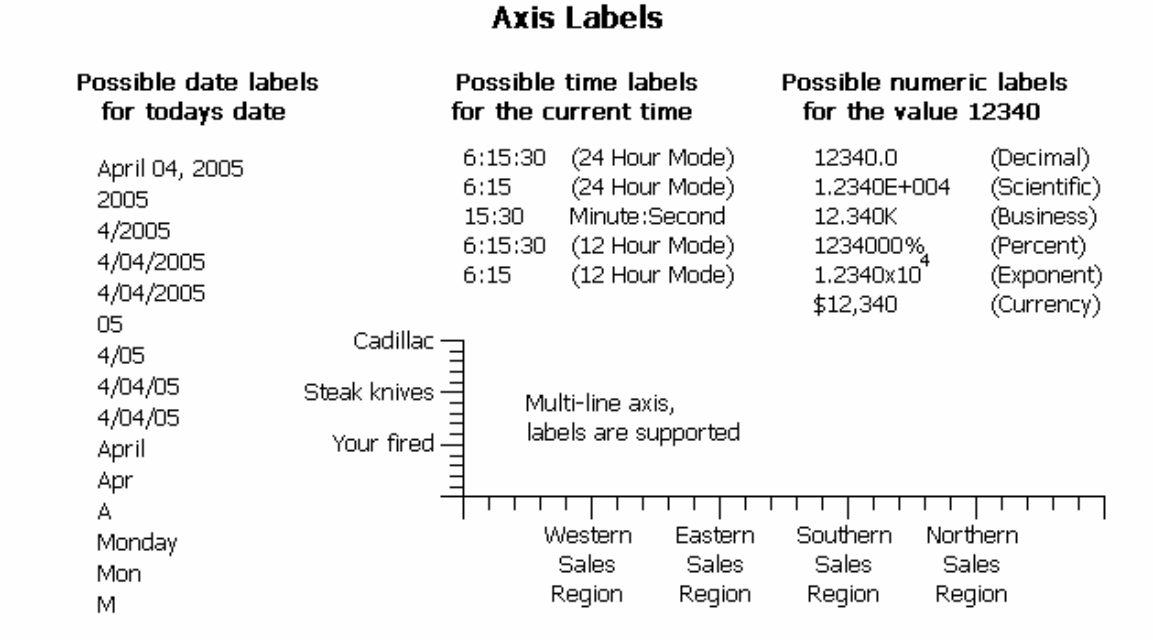
Axis Label Classes

AxisLabels

- **NumericAxisLabels**
- **StringAxisLabels**
- **PolarAxesLabels**
- **AntennaAxesLabels**
- **TimeAxisLabels**

ElapsedTimeAxisLabels

Axis labels inform the user of the x- and y-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.



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)..

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.

ElapsedTimeAxisLabels This class labels the major tick marks of the associated **ElapsedTimeAxis** object. The class supports HH:MM:SS and MM:SS formats, with decimal seconds out to 0.00001, i.e. “12:22:43.01234”. It also supports a cumulative hour format (101:51:22), and a couple of day formats (4.5:51:22, 4D 5:51:22).

PolarAxesLabels This class labels the major tick marks of the associated **PolarAxes** object. The x-axis is labeled from 0.0 to the polar scale magnitude, and the circular axis is labeled counter clockwise from 0 to 360 degrees, starting at 3:00.

AntennaAxesLabels This class labels the major tick marks of the associated **AntennaAxes** object. The y-axis is labeled from the radius minimum to the radius maximum. The circular axis is labeled clockwise from 0 to 360 degrees, starting at 12:00.

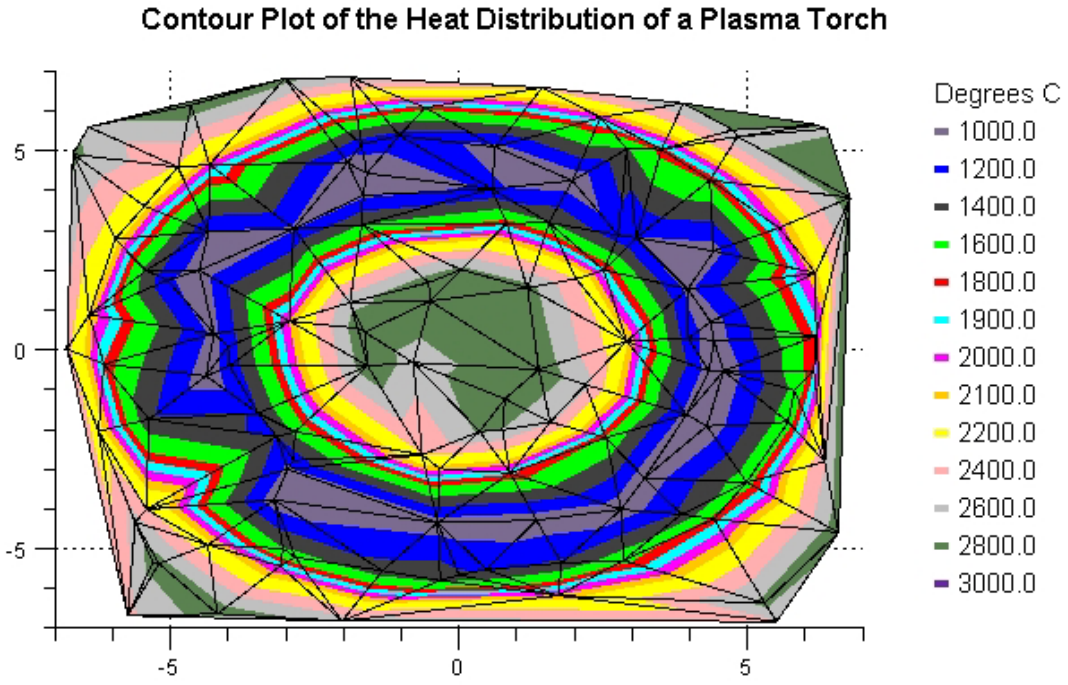
Chart Plot Classes

ChartPlot
- **ContourPlot**
- **GroupPlot**
- **PieChart**
- **PolarPlot**
- **AntennaPlot**
- **SimplePlot**

Plot objects are objects that display data organized in a **ChartDataset** class. There are six main categories: simple, group, polar, antenna, contour and pie plots. Simple plots graph data organized as a simple set of xy data points. The most common examples of simple plots are line plots, bar graphs, scatter plots and line-marker plots. Group plots graph data organized as multiple y-values for each x-value. The most common examples of group plots are stacked bar graphs, open-high-low-close plots, candlestick plots, floating stacked bar plots and “box and whisker” plots. Polar charts plot data organized as a simple set of data points, where each data point represents a polar magnitude and angle pair, rather than xy Cartesian coordinate values. The most common example of polar charts is the display of complex numbers (a + b*i*), and it is used in many engineering disciplines. Antenna charts plot data organized as a simple set of data points, where each data point represents a radius value and angle pair, rather than xy Cartesian coordinate

values. The most common example of antenna charts is the display of antenna performance and specification graphs. 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, were 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.



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

Group Plot Classes

GroupPlot

ArrowPlot

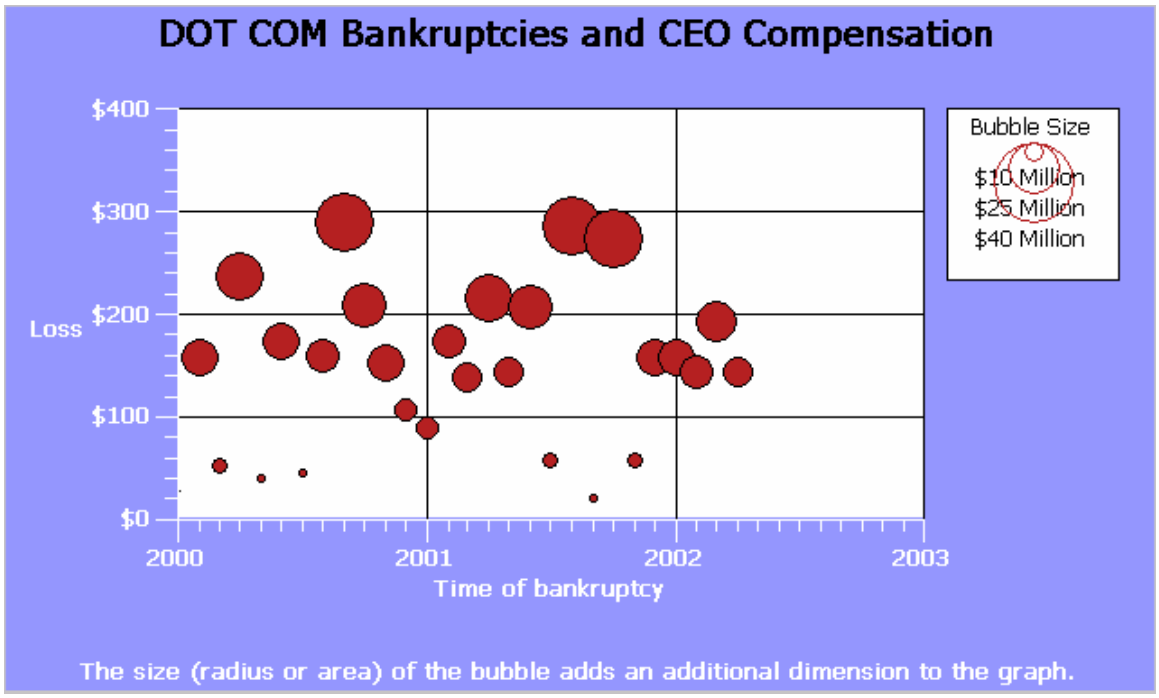
BoxWhiskerPlot

BubblePlot
CandlestickPlot
CellPlot
ErrorBarPlot
FloatingBarPlot
FloatingStackedBarPlot
GroupBarPlot
GroupVersaPlot
HistogramPlot
LineGapPlot
MultiLinePlot
OHLCPlot
StackedBarPlot
StackedLinePlot
GroupVeraPlot

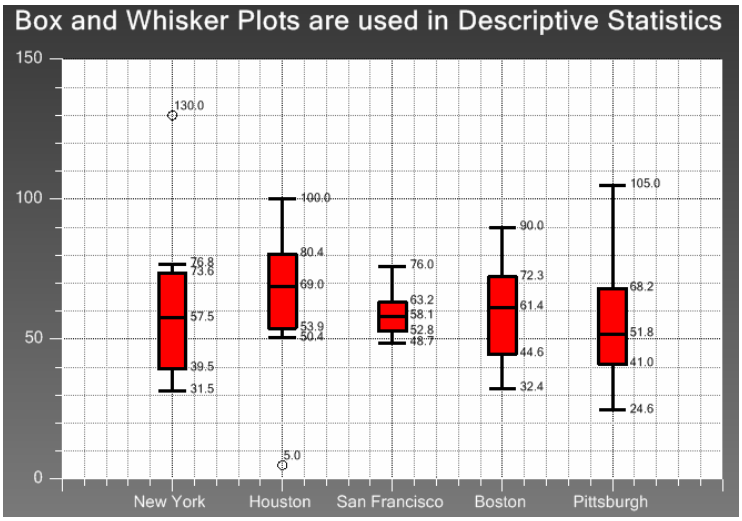
Group plots use data organized as arrays of x- and y-values, where there is one or more y for every x.. Group plot types include multi-line plots, stacked line plots, stacked bar plots, group bar plots, error bar plots, floating bar plots, floating stacked bar plots, open-high-low-close plots, candlestick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

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

ArrowPlot This class is a concrete implementation of the **GroupPlot** class and 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

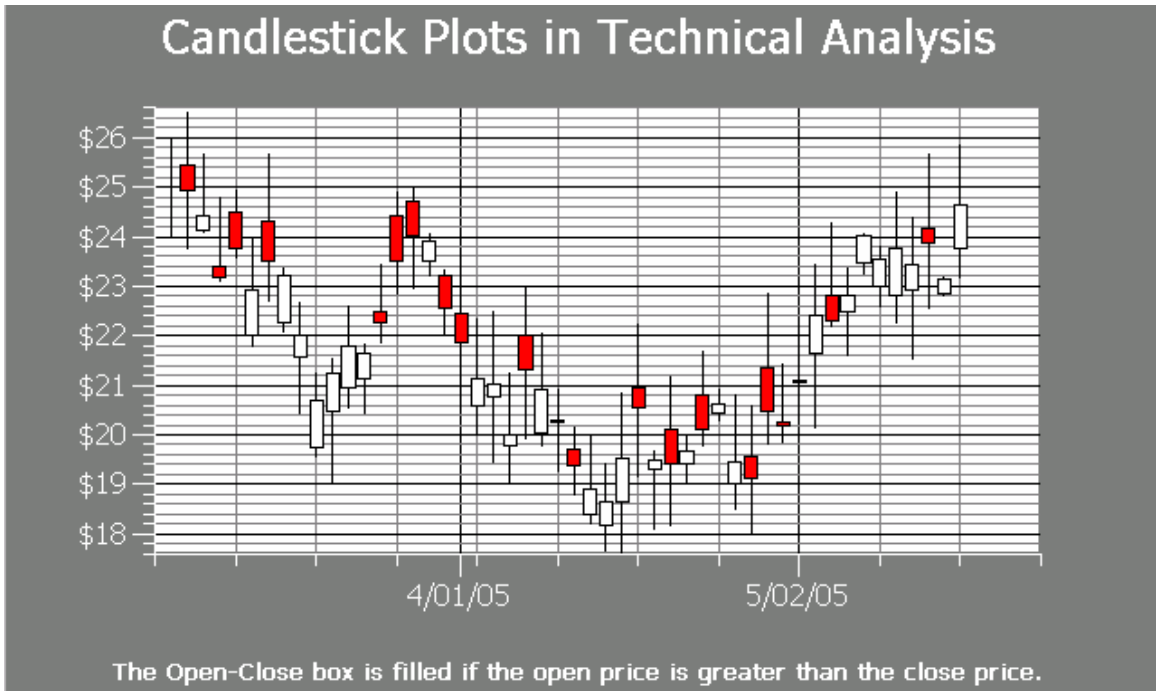


BubblePlot This class is a concrete implementation of the **GroupPlot** class and displays bubble plots. The values in the dataset specify the position and size of each bubble in a bubble chart.

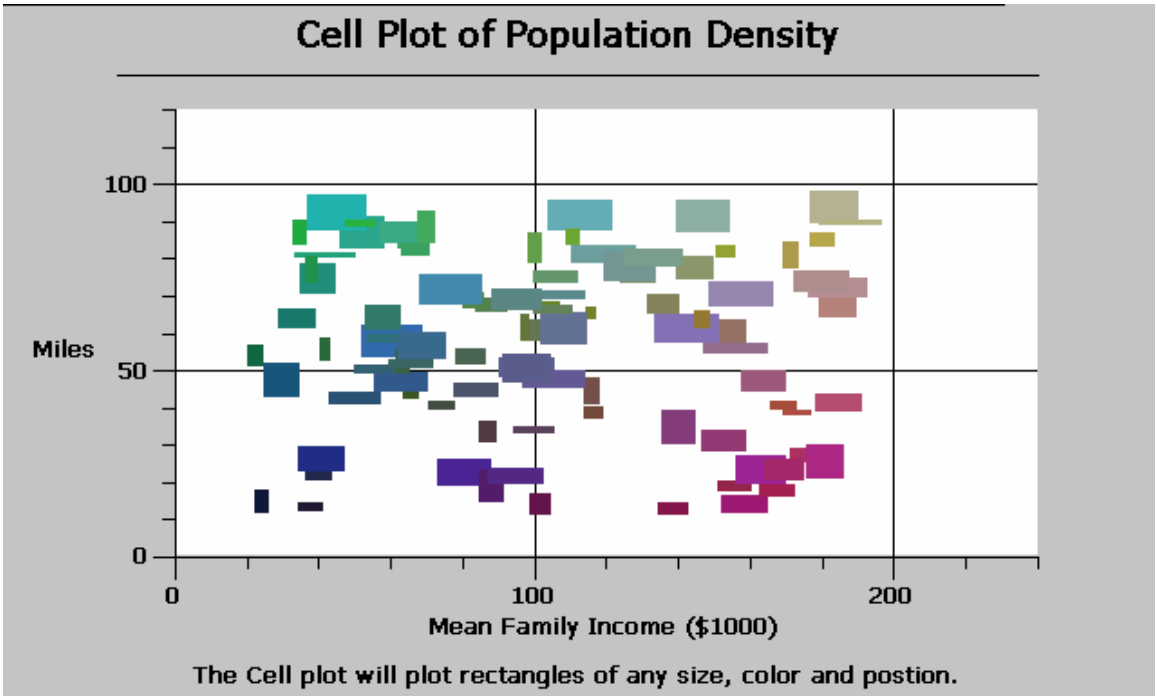


BoxWhiskerPlot This class is a concrete implementation of the **GroupPlot** class and displays box and whisker plots. The BoxWhiskerPlot class graphically depicts groups of numerical data through their five-number summaries (the

smallest observation, lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation).

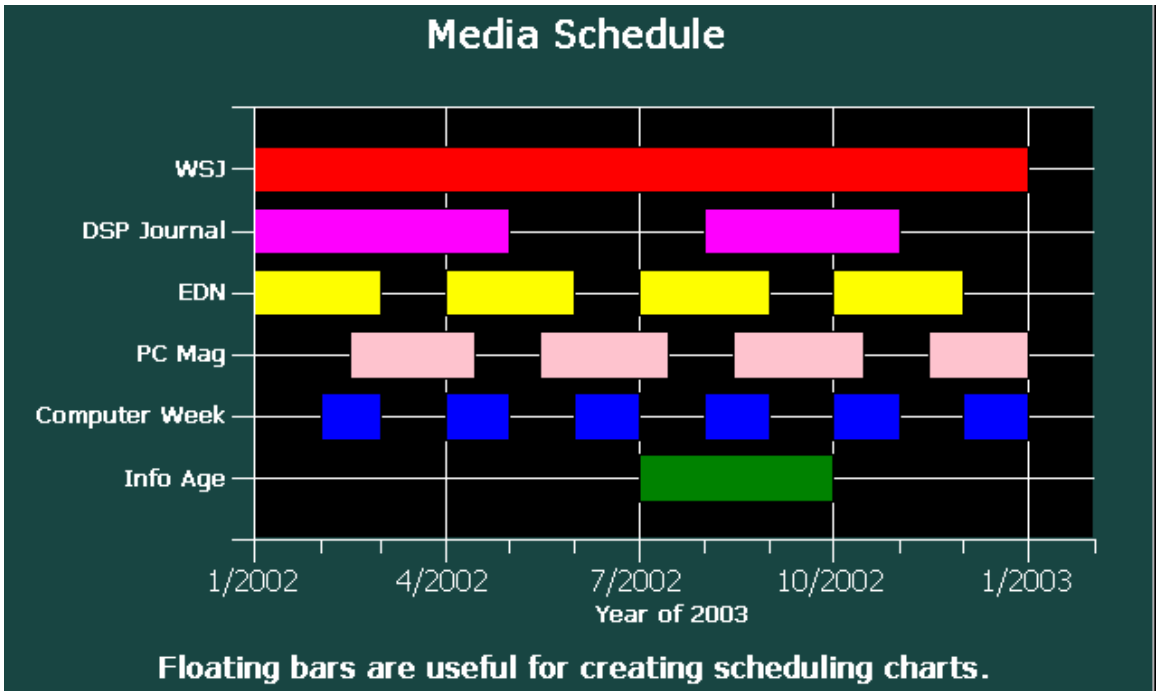


CandlestickPlot This class is a concrete implementation of the **GroupPlot** class and displays stock market data in an open-high-low-close format common in financial technical analysis.

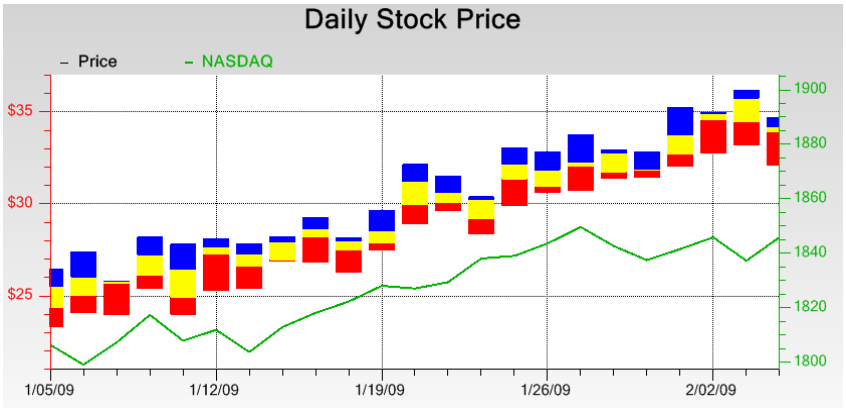


CellPlot This class is a concrete implementation of the **GroupPlot** class and displays cell plots. A cell plot is a collection of rectangular objects with independent positions, widths and heights, specified using the values of the associated group dataset.

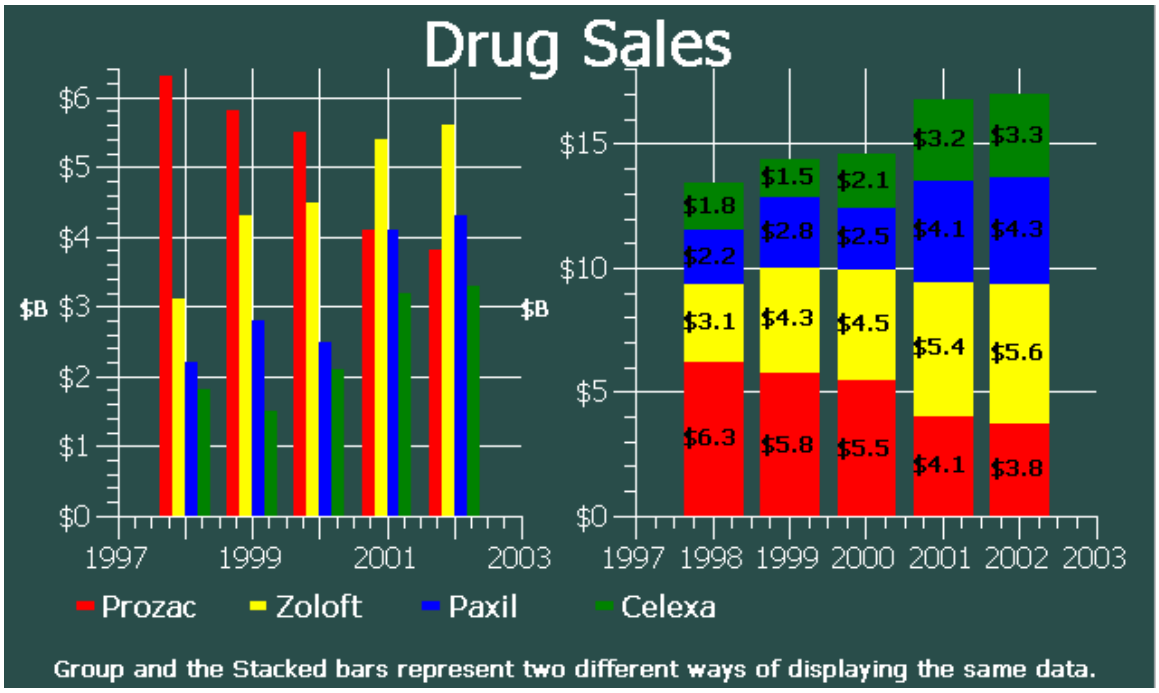
ErrorBarPlot This class is a concrete implementation of the **GroupPlot** class and displays error bars. Error bars are two lines positioned about a data point that signify the statistical error associated with the data point



FloatingBarPlot This class is a concrete implementation of the **GroupPlot** class and displays free-floating bars in a graph. The bars 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.



FloatingStackedBarPlot This class is a concrete implementation of the **GroupPlot** class and displays free-floating stacked bars. The bars 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.



GroupBarPlot

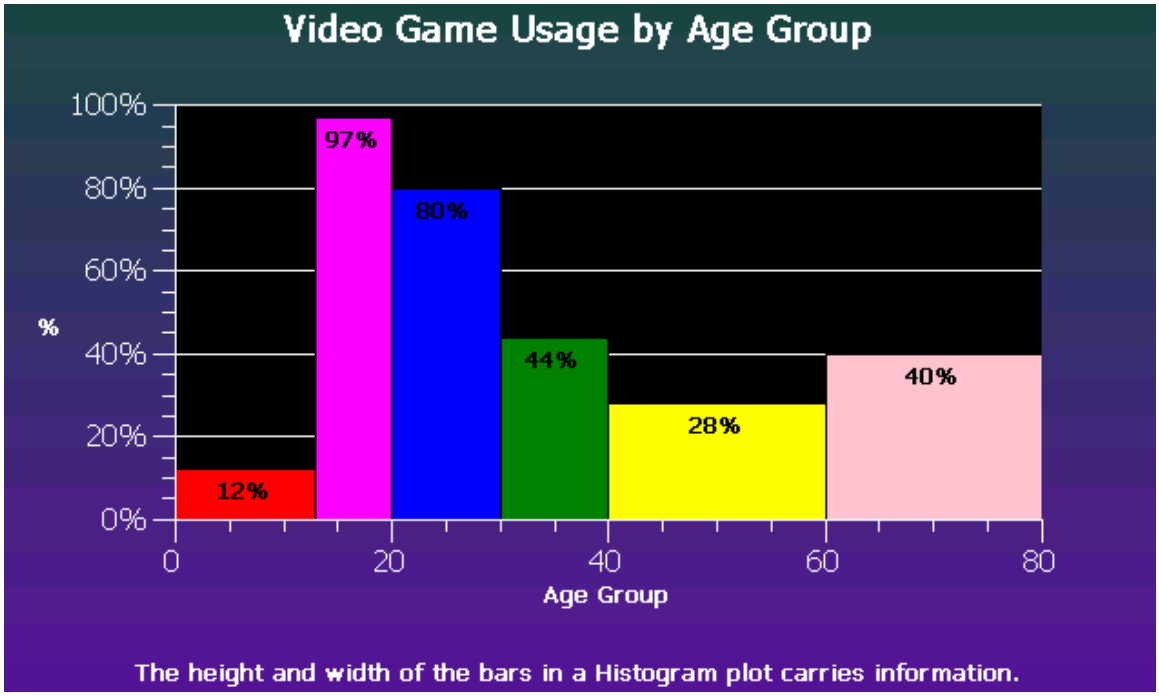
This class is a concrete implementation of the **GroupPlot** class and displays group data in a group bar format. Individual bars, the height of which corresponds to the group y-values of the dataset, display side by side, as a group, justified with respect to the x-position value for each group. The group bars share a common base value.

StackedBarPlot

This class is a concrete implementation of the **GroupPlot** class and displays 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.

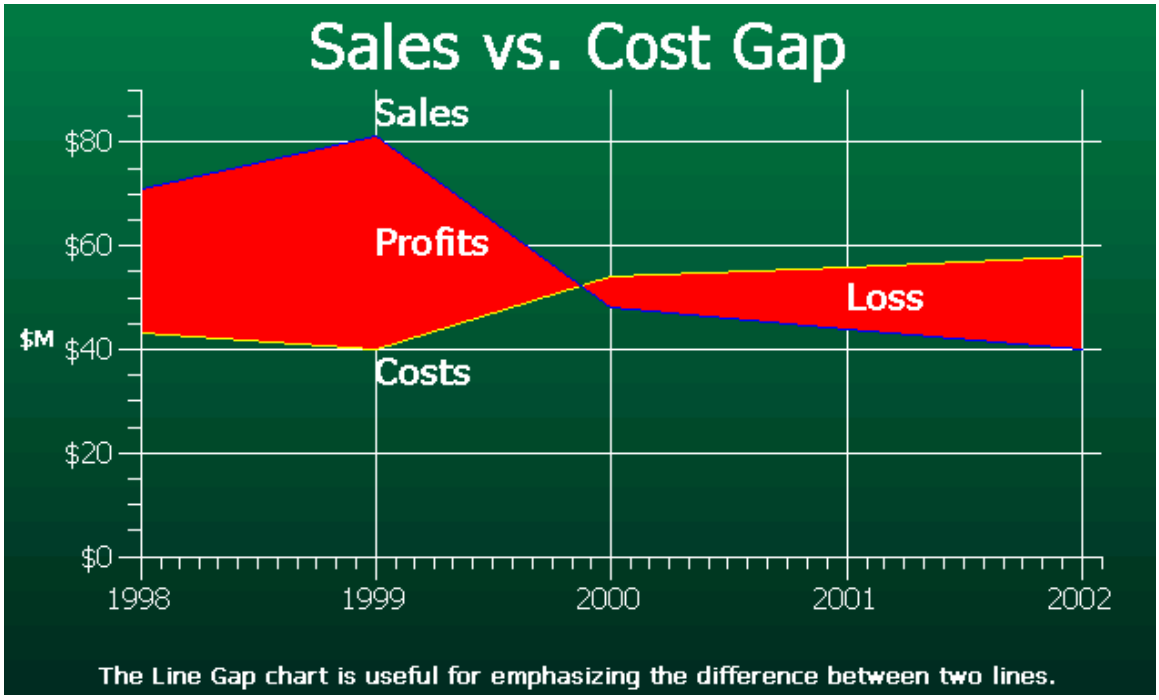
GroupVeraPlot

The **GroupVersaPlot** is a plot type that can be any of the eight group plot types: GROUPBAR, STACKEDBAR, CANDLESTICK, OHLC, MULTILINE, STACKEDLINE, FLOATINGBAR and FLOATING_STACKED_BAR. Use it when you want to be able to change from one plot type to another, without deleting the instance of the old plot object and creating an instance of the new.

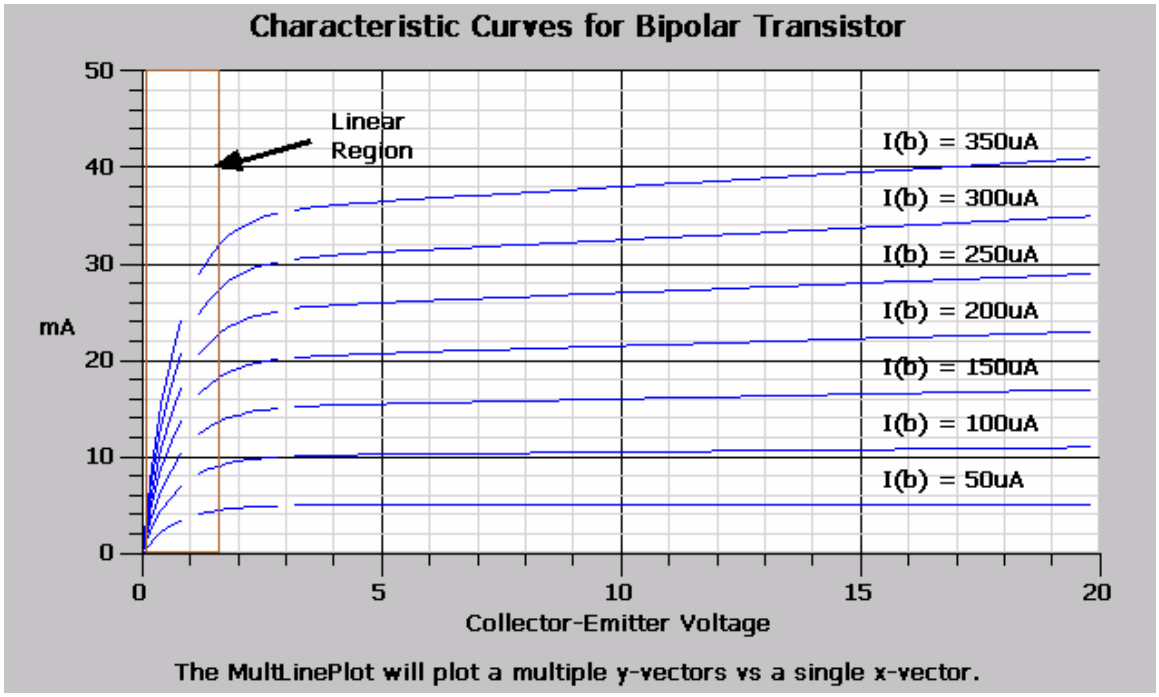


HistogramPlot

This class is a concrete implementation of the **GroupPlot** class and displays 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.

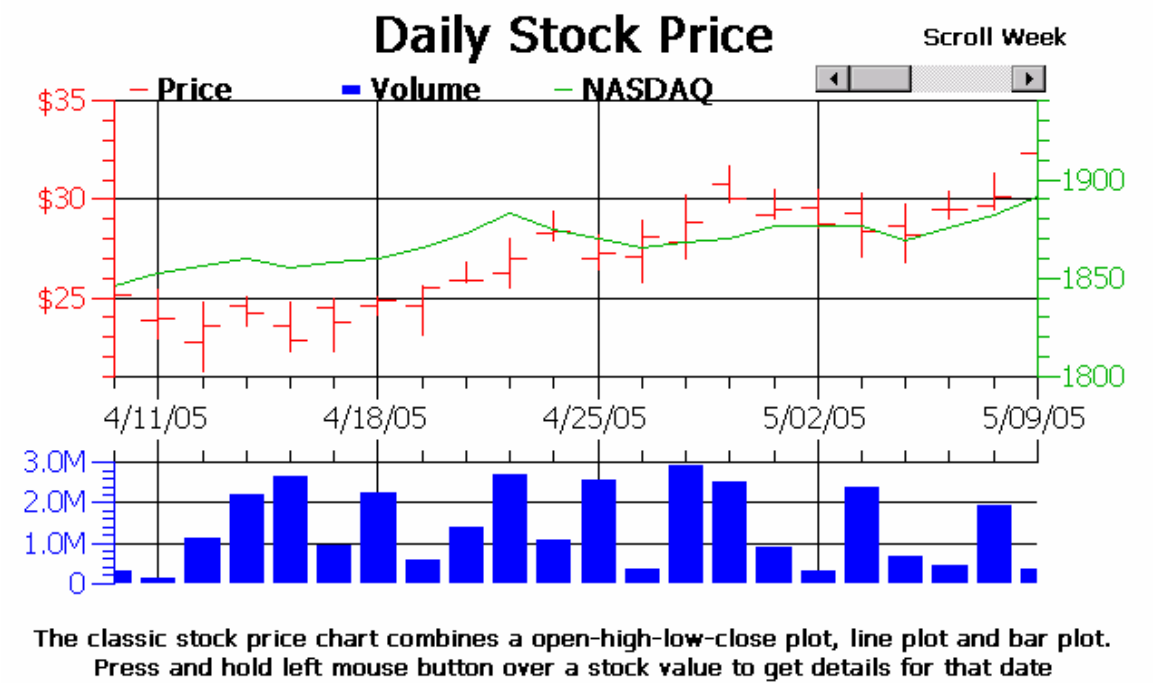


LineGapPlot This class is a concrete implementation of the **GroupPlot** class. A line gap chart consists of two lines plots where a contrasting color fills the area between the two lines, highlighting the difference.



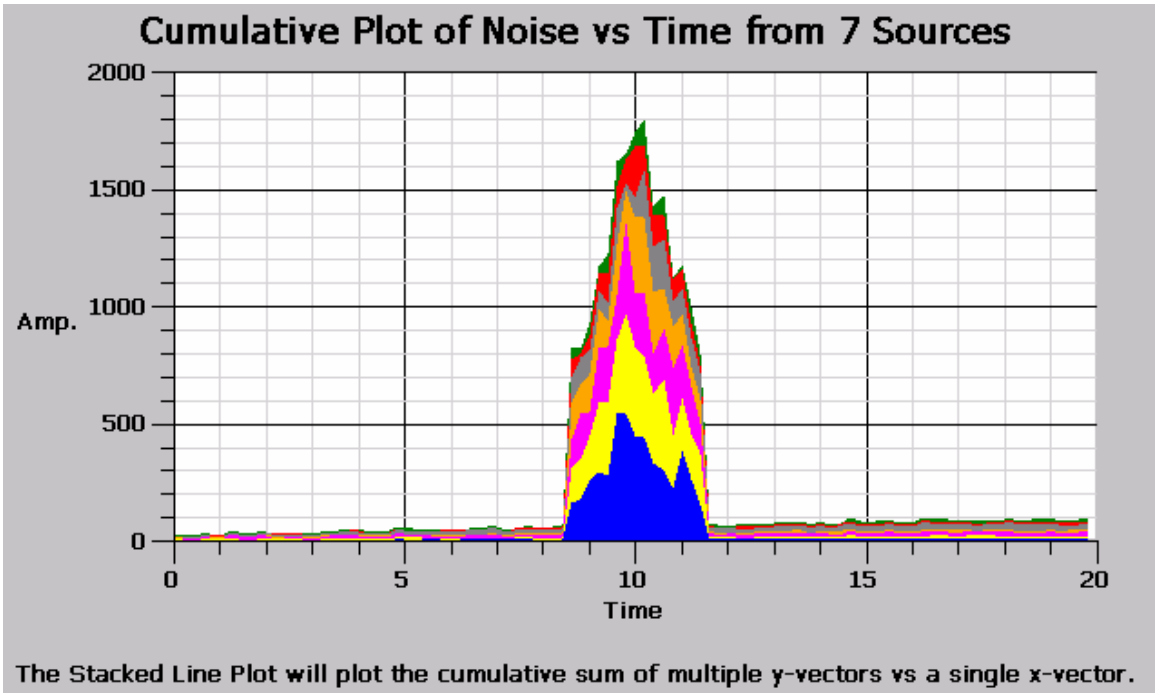
MultiLinePlot

This class is a concrete implementation of the **GroupPlot** class and 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.



OHLCPlot

This class is a concrete implementation of 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.



StackedLinePlot This class is a concrete implementation of the **GroupPlot** class and 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.

Polar Plot Classes

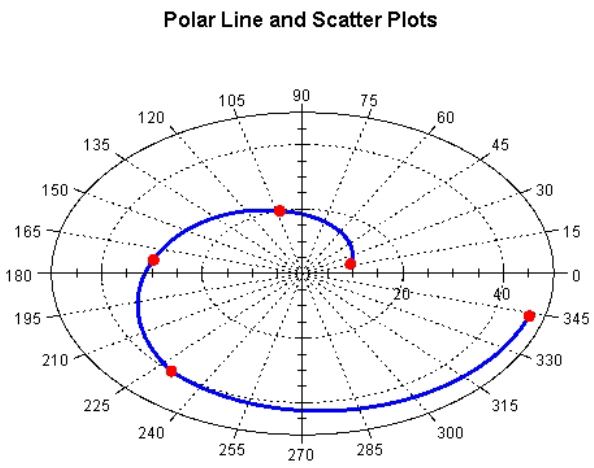
PolarPlot

PolarLinePlot

PolarScatterPlot

Polar plots that use data organized as arrays of x- and y-values, where an x-value represents the magnitude of a point in polar coordinates, and the y-value represents the angle, in radians, of a point in polar coordinates. Polar plot types include line plots and scatter plots.

PolarPlot This class is an abstract base class for the polar plot classes.



PolarLinePlot This class is a concrete implementation of the **PolarPlot** class and displays data in a simple line plot format. The lines drawn between adjacent data points use polar coordinate interpolation.

PolarScatterPlot This class is a concrete implementation of the **PolarPlot** class and displays data in a simple scatter plot format.

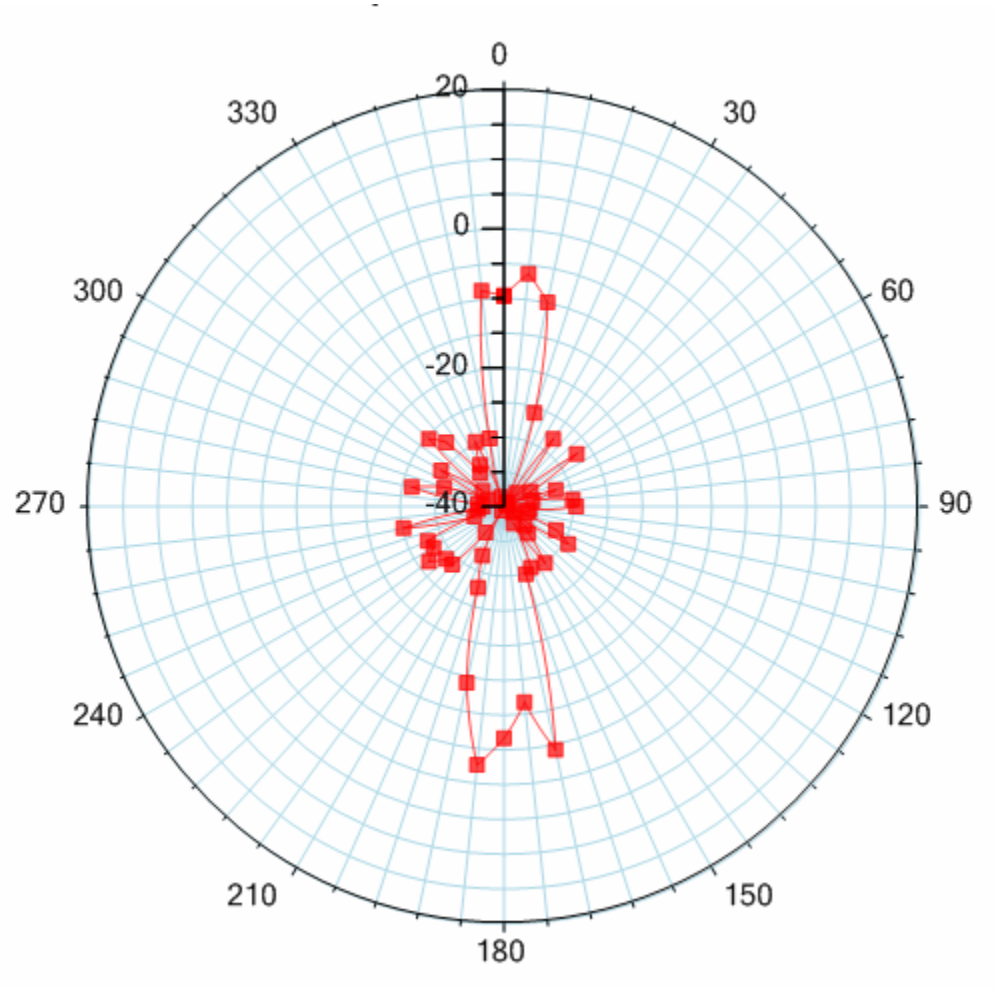
Antenna Plot Classes

AntennaPlot
- **AntennaLinePlot**
- **AntennaScatterPlot**
- **AntennaLineMarkerPlot**

GraphObj
- **AntennaAnnotation**

Antenna plots that use data organized as arrays of x- and y-values, where an x-value represents the radial value of a point in antenna coordinates, and the y-value represents the angle, in degrees, of a point in antenna coordinates. Antenna plot types include line plots, scatter plots, line marker plots, and an annotation class.

AntennaPlot This class is an abstract base class for the polar plot classes.



AntennaLineMarkerPlot

AntennaLinePlot This class is a concrete implementation of the **AntennaPlot** class and displays data in a simple line plot format. The lines drawn between adjacent data points use antenna coordinate interpolation.

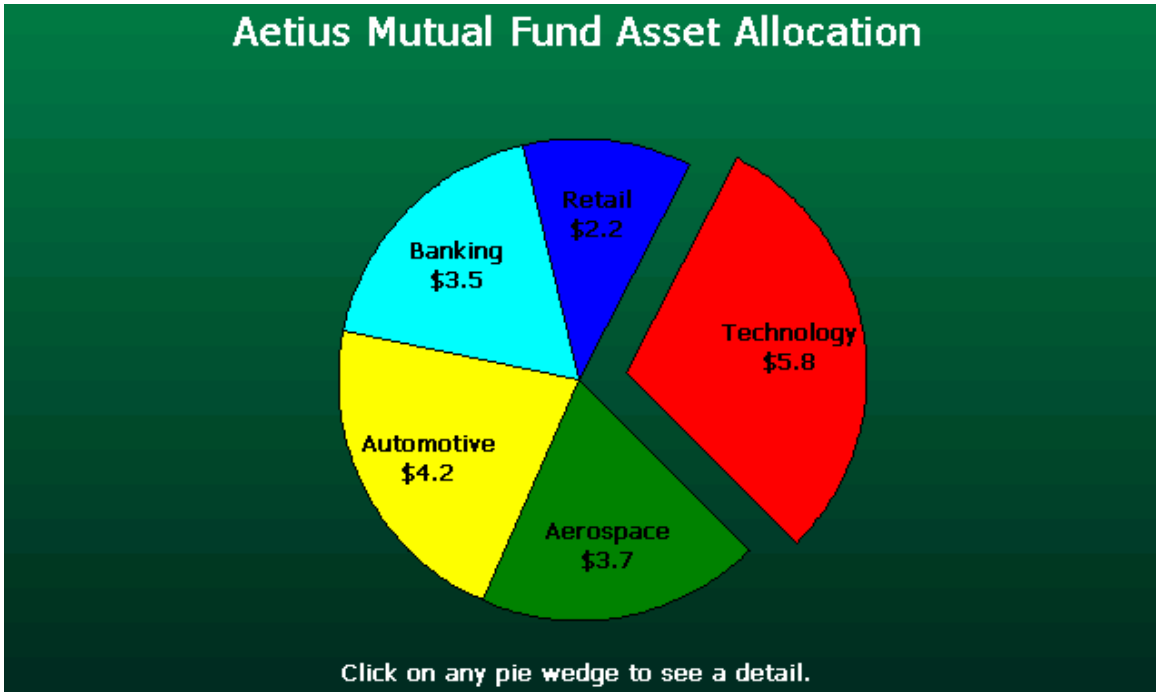
AntennaScatterPlot This class is a concrete implementation of the **AntennaPlot** class and displays data in a simple scatter plot format.

AntennaLineMarkerPlot This class is a concrete implementation of the **AntennaPlot** class and displays data in a simple line markder plot format.

AntennaAnnotation This class is used to highlight, or mark, a specific attribute of the chart. It can mark a constant radial value using a circle, or it can mark a constant angular value using a radial line from the origin to the outer edge of the scale.

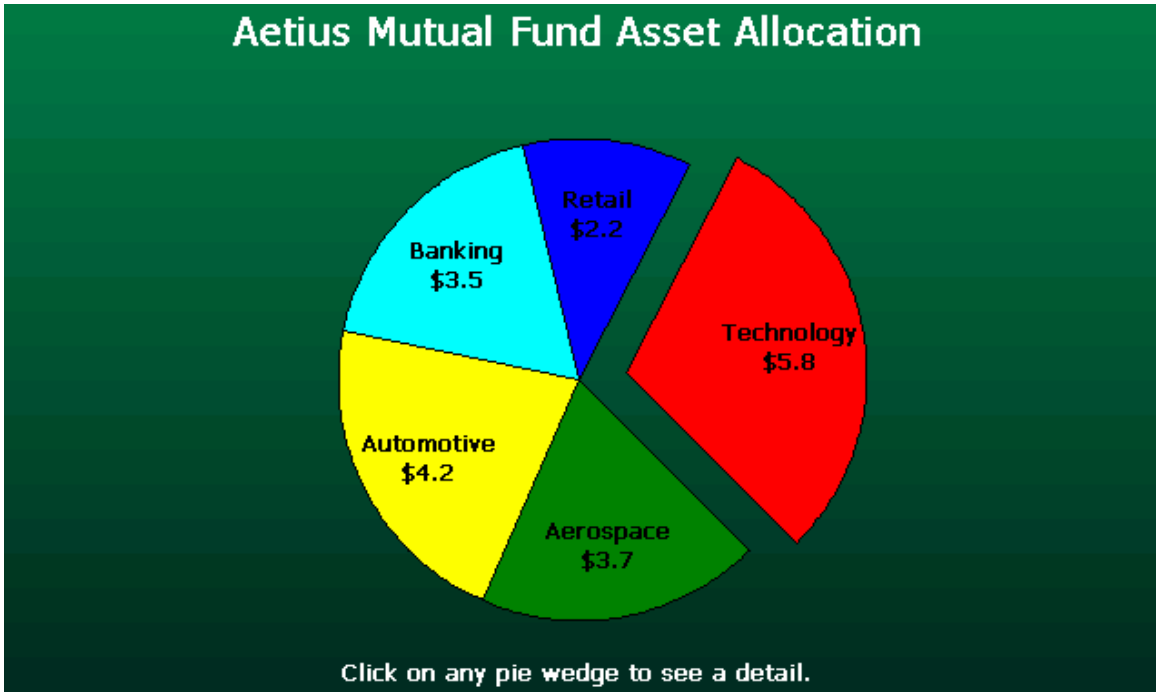
Pie and Ring Chart Classes

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.



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.



RingChart

The ring chart plots data in a modified pie chart format known as a ring chartt. It uses data organized as arrays of x- and y-values, where an x-value represents the numeric value of a ring segment, and a y-value specifies the offset (or "explosion") of a ring segment with respect to the origin of the ring.

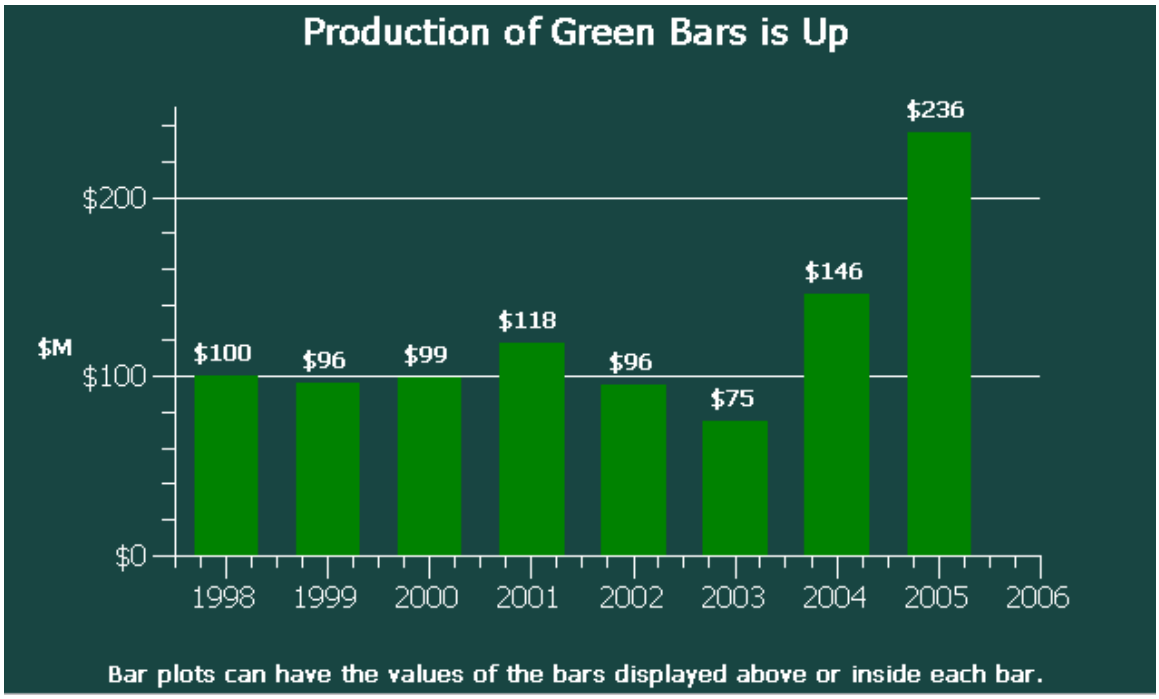
Simple Plot Classes

SimplePlot

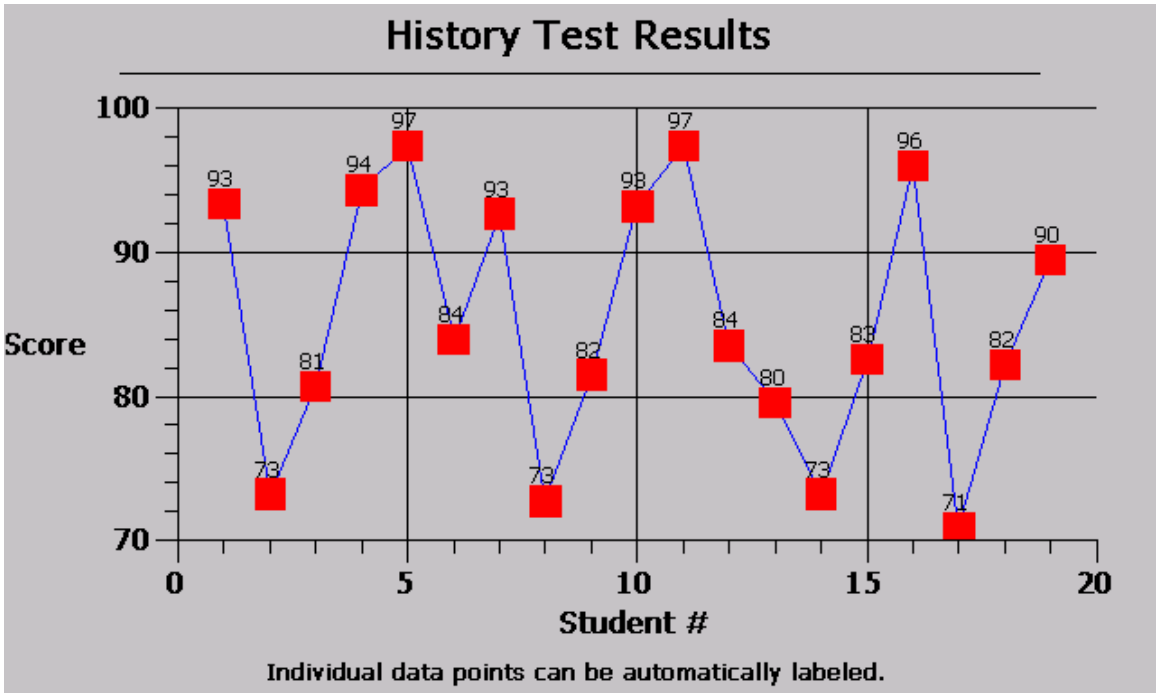
- **SimpleBarPlot**
- **SimpleLineMarkerPlot**
- **SimpleLinePlot**
- **SimpleScatterPlot**
- **SimpleVeraPlot**

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.

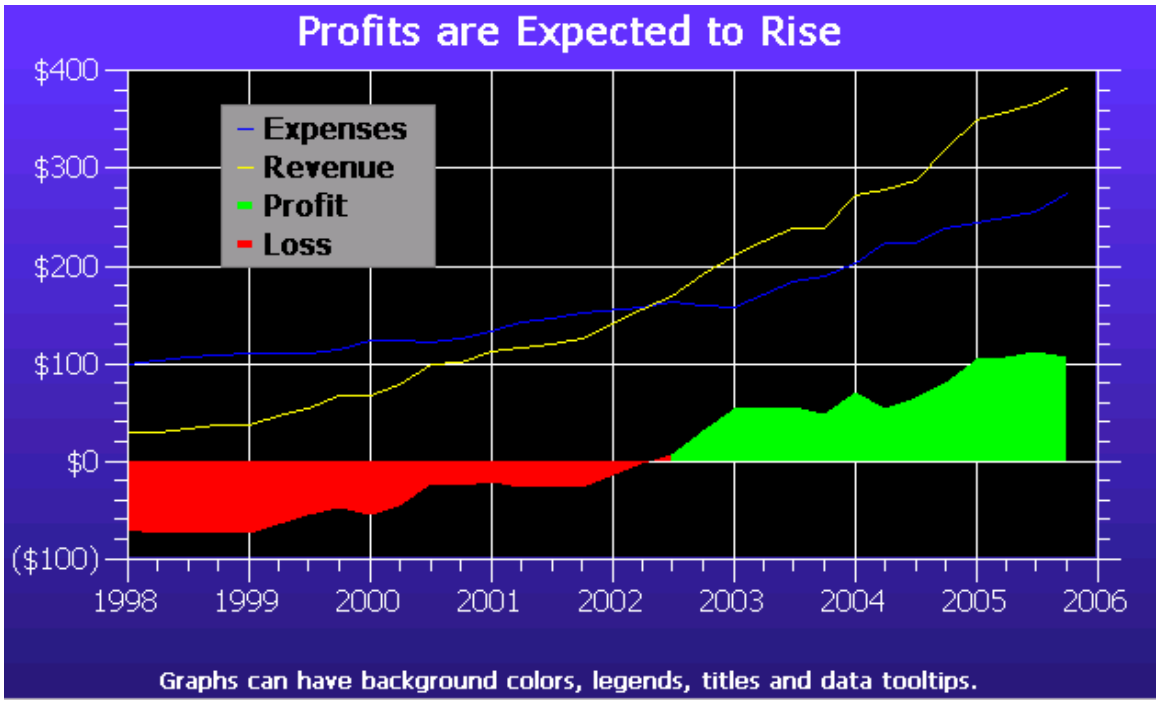


SimpleBarPlot This class is a concrete implementation of the **SimplePlot** class and displays data in a bar format. Individual bars, the maximum value of which corresponds to the y-values of the dataset, are justified with respect to the x-values.

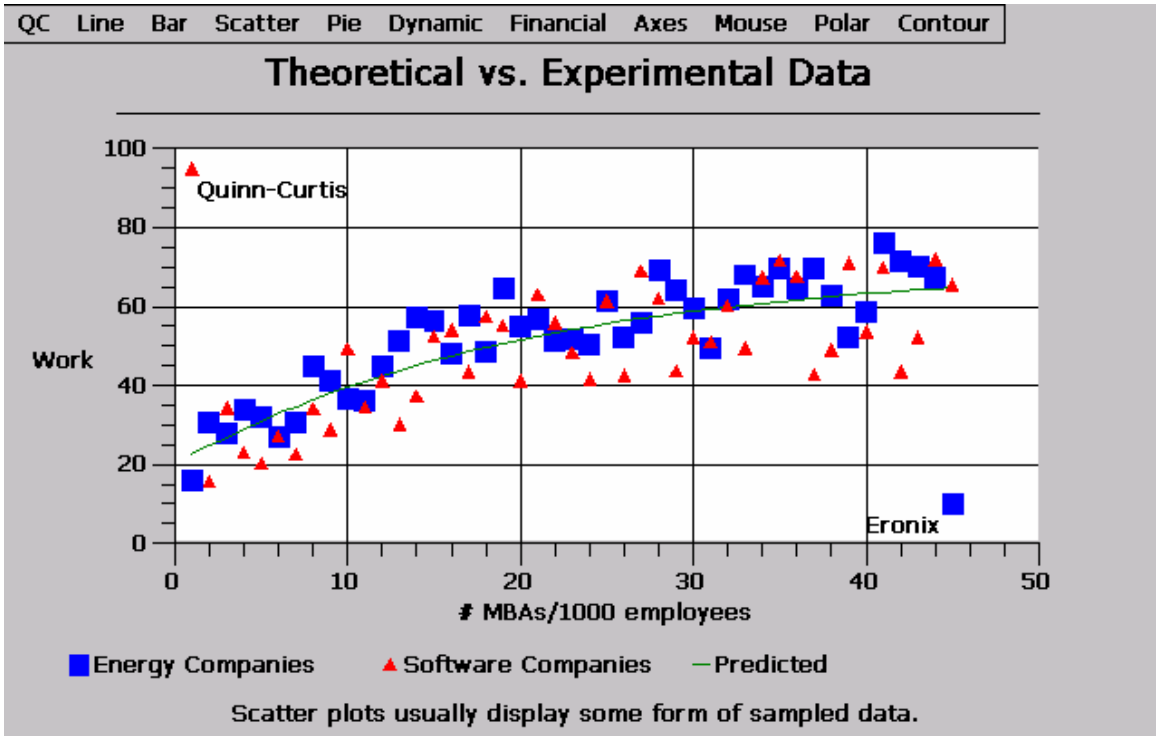


SimpleLineMarkerPlot

This class is a concrete implementation of the **SimplePlot** class and it displays simple datasets in a line plot format where scatter plot symbols highlight individual data points.



SimpleLinePlot This class is a concrete implementation 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.



SimpleScatterPlot This class is a concrete implementation of the **SimplePlot** class and it displays simple datasets in a scatter plot format where each data point is represented using a symbol.

SimpleVersaPlot The **SimpleVersaPlot** is a plot type that can be any of the four simple plot types: LINE_MARKER_PLOT, LINE_PLOT, BAR_PLOT, SCATTER_PLOT. It is used when you want to be able to change from one plot type to another, without deleting the instance of the old plot object and creating an instance of the new.

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 a concrete implementation of the **Legend** class and it 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 concrete implementation of the **Legend** class and it 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

PolarGrid
AntennaGrid

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 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 of the chart. This class works in conjunction with the **LinearAxis**, **LogAxis** and **TimeAxis** classes.

PolarGrid This class defines the grid lines associated with a polar axis. A polar chart grid consists of two sets of lines. The first set is a group of concentric circles, centered on the origin and passing through the major and/or minor tick marks of the polar magnitude horizontal and vertical axes. The second set is a group of radial lines, starting at the origin and extending to the outermost edge of the polar plot circle, passing through the major and minor tick marks of the polar angle circular axis. This class works in conjunction with the **PolarAxes** class.

AntennaGrid Analogous to the **PolarGrid**, this class draws radial, and circular grid lines for an Antenna chart.

Chart Text Classes

ChartText
- **ChartTitle**
- **AxisTitle**
- **ChartLabel**
  - **NumericLabel**
  - **TimeLabel**
  - **StringLabel**
  - **ElapsedTimeLabel**

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 concrete implementation of the **ChartLabel** class and it displays formatted numeric values.
TimeLabel	This class is a concrete implementation of the **ChartLabel** class and it displays formatted **ChartCalendar** dates.
ElapsedTimeLabel	This class is a concrete implementation of the **ChartLabel** class and it displays numeric values formatted as elapsed time strings **(12:32:21).**
StringLabel	This class is a concrete implementation of the **ChartLabel** class that formats string values for use as axis labels.

Miscellaneous Chart Classes

Marker
ChartImage
ChartShape
ChartSymbol

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.
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.

ChartShape This class encapsulates a **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.**GraphicsPath** class.

ChartSymbol This class defines symbols used by the **SimplePlot** scatter plot functions. Pre-defined symbols include square, triangle, diamond, cross, plus, star, line, horizontal bar, vertical bar, 3D bar and circle.

Mouse Interaction Classes

MouseListener
- **MoveObj**
- **FindObj**
- **DataToolTip**
- **DataCursor**
  - **MoveData**
- **MagniView**
- **MoveCoordinates**
- **MultiMouseListener**
- **ChartZoom**

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, ChartZoom, MagniView and MoveCoordinates** classes also implement mouse event delegates that use the mouse to mark, move and zoom chart objects and data.

MouseListener This class implements .Net CF 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, group values and open-high-low-close values, for a given point in a chart.

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 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

MagniView This class implements mouse controlled magnification for one or more simultaneous axes. This class implements a chart magnify class based on the **MouseListener** class. It uses two charts; the source chart and the target chart. The source chart displays the chart in its unmagnified state. The target chart displays the chart in the magnified state. The mouse positions a **MagniView** rectangle within the source chart, and the target chart is re-scaled and redrawn to

match the extents of the **MagniView** rectangle from the source chart.

MoveCoordinates This class extends the **MouseListener** class and it can move the coordinate system of the underlying chart, analogous to moving (chaging the coordinates of) an internet map by "grabbing" it with the mouse and dragging it.

MultiMouseListener This class is used by the **ChartView** class to support multiple mouse listeners at the same time.

Miscellaneous Utility Classes

ChartCalendar
CSV
Dimension
Point2D
GroupPoint2D
DoubleArray
DoubleArray2D
BoolArray
Point3D
NearestPointData
TickMark
Polysurface
Rectangle2D

ChartCalendar This class contains utility routines used to process **ChartCalendar** date objects.

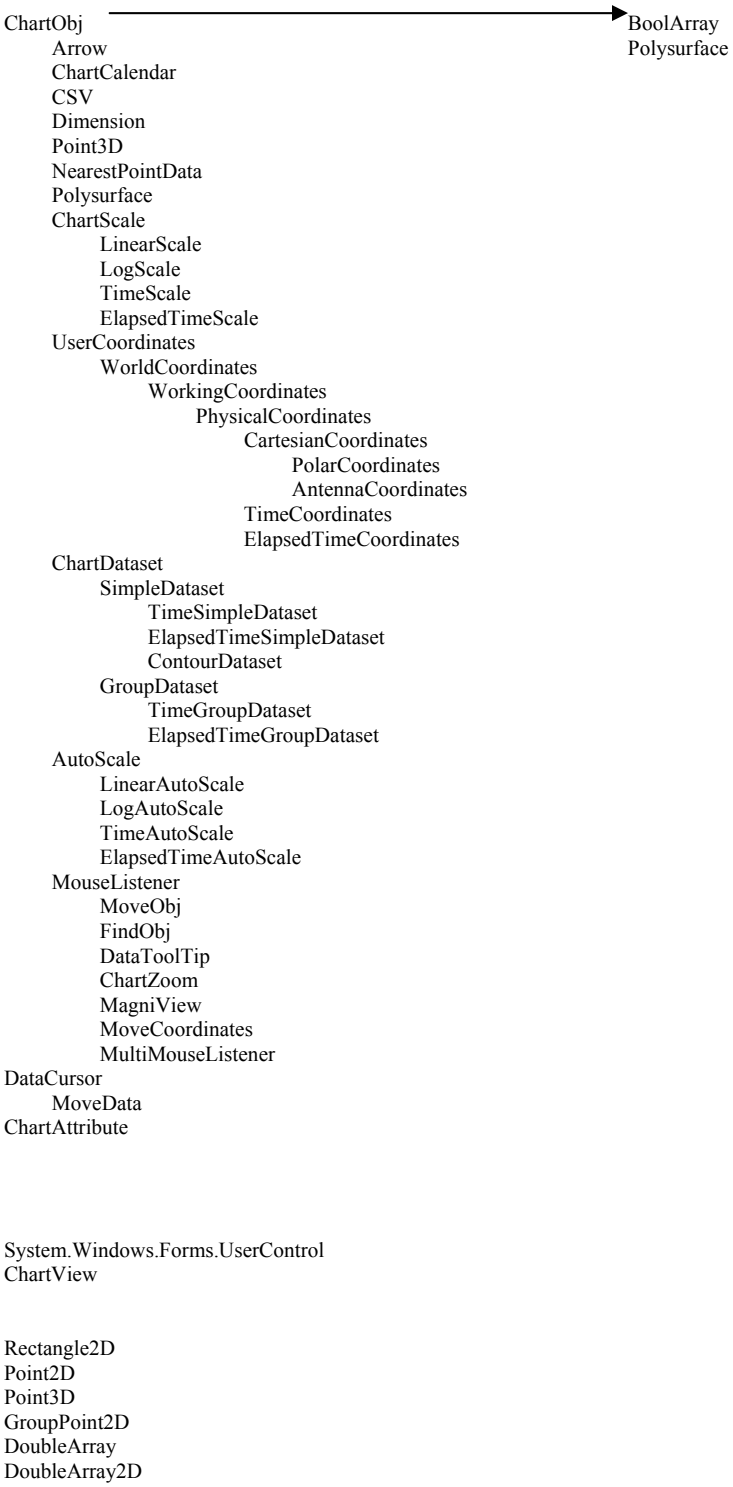
CSV This is a utility class for reading and writing CSV (Comma Separated Values) files.

Dimension This is a utility class for handling dimension (height and width) information using doubles, rather than the integers used by the Size class.

Point2D This class encapsulates an xy pair of values as doubles (more useful in this software than the .Net CF Point class).

GroupPoint2D	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.
DoubleArray	This class is used as an alternative to the standard .Net CF 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 CF 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 CF Array class, adding routines for resizing of the array, and the insertion and deletion of bool based data elements.
Point3D	This class encapsulates an xyz set of double values used to specify 3D data values.
NearestPointData	This is a utility class for returning data that results from nearest point calculations.
TickMark	The axis classes use this class 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	This is a utility class that extends the RectangleF class, using doubles as internal storage.

A diagram depicts the class hierarchy of the QCChart2D CF for the .Net Compact Framework library.



- GraphObj
  - TickMark
  - Axis
    - LinearAxis
      - PolarAxes
      - AntennaAxes
    - LogAxis
    - TimeAxis
    - ElapsedTimeAxis
  - ChartText
    - ChartTitle
    - AxisTitle
    - ChartLabel
      - NumericLabel
        - BarDatapointValue
      - TimeLabel
      - ElapsedTimeLabel
      - StringLabel
    - AxisLabels
      - NumericAxisLabels
      - TimeAxisLabels
      - ElapsedTimeAxisLabels
      - StringAxisLabels
      - PolarAxesLabels
      - AntennaAxesLabels
  - Grid
    - PolarGrid
    - AntennaGrid
  - LegendItem
  - BubblePlotLegendItem
  - Legend
    - StandardLegend
    - BubblePlotLegend
  - ChartPlot
    - SimplePlot
      - SimpleLinePlot
      - SimpleBarPlot
      - SimpleScatterPlot
      - SimpleLineMarkerPlot
      - SimpleVersaPlot
    - GroupPlot
      - ArrowPlot
      - BubblePlot
      - CandlestickPlot
      - CellPlot
      - ErrorBarPlot
      - FloatingBarPlot
      - FloatingStackedBarPlot
      - GroupBarPlot
      - HistogramPlot
      - LineGapPlot
      - MultiLinePlot
      - OHLCPlot
      - StackedBarPlot
      - StackedLinePlot
      - BoxWhiskerPlot
      - GroupVersaPlot
    - PieChart
    - RingChart
    - PolarPlot
      - PolarLinePlot
      - PolarScatterPlot
    - AntennaPlot
      - AntennaLinePlot
      - AntennaScatterPlot
      - AntennaLineMarkerPlot
- Background
- ChartImage
- ChartShape
- ChartSymbol
- Marker
- ChartZoom

3. Chart Datasets

ChartDataset
- **SimpleDataset**
  - **TimeSimpleDataset**
  - **ElapsedTimeSimpleDataset**
  - **ContourDataset**
- **GroupDataset**
  - **TimeGroupDataset**
  - **ElapsedTimeGroupDataset**

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, or **TimeSpan** as the x- or y-values. Values. Also, there is a dataset type that is used to plot contour data.

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, inititialize 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.

Simple Numeric Dataset

Class SimpleDataset

ChartObj
- |
- **+--ChartDataset**
  - |

+--SimpleDataset

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]

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- and 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() _
)


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

sname	Specifies the name of the dataset.
x	An array that specifies the x-values of a dataset.
y	An array that specifies the y-values of a dataset. The length of the y array must match the length of the x array.

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
);
```

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 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 **Point2D** 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 Dataset1 As SimpleDataset = New SimpleDataset("First", x1, y1)

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
 Dim Dataset2 As SimpleDataset = New SimpleDataset("Second", x2, y2)
```

[C#]

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

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;
SimpleDataset Dataset2 = new SimpleDataset("Second", x2, y2);
```

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.WriteSimpleDataset(csvdata,"SimpleDataset2.csv");
```

[Visual Basic]

```
Dim csvdata As CSV = New CSV()
Dim Dataset1 As SimpleDataset =
       New SimpleDataset(csvdata, "SimpleDataset.csv", 0, 0)
' 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);
Point2D 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 Point2D = 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
```

Simple Date/Time Dataset

Class TimeSimpleDataset

ChartObj
|
+--ChartDataset
|
+--SimpleDataset
|
+-TimeSimpleDataset

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

sname	Specifies the name of the dataset.
x	An array that specifies the **ChartCalendar** x-values of a dataset.
y	An array that specifies the y-values of a dataset. The length of the y array must match the length of the x array.

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 **SetDateTimeFomatString** method to change the default date time format. See the documentation for the .Net CF **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 CF 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 nnumpnts = 32;
ChartCalendar [] x1= new ChartCalendar[nnumpnts];
double []y1 = new double[nnumpnts];
double []y2 = new double[nnumpnts];
y1[0] = 100;
y2[0] = 30;
x1[0] = (ChartCalendar) currentdate.Clone();
currentdate.Add(ChartObj.MONTH,3);
for (i=1; i < nnumpnts; 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);
}
TimeSimpleDataset Dataset1 = new TimeSimpleDataset("Sales",x1,y1);
TimeSimpleDataset Dataset2 = new TimeSimpleDataset("Expenses",x1,y2);
```

[Visual Basic]

```
Dim nnumpnts As Integer = 32
Dim x1(nnumpnts-1) As ChartCalendar
Dim y1(nnumpnts-1) As Double
Dim y2(nnumpnts-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 nnumpnts - 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
Dim Dataset1 As New TimeSimpleDataset("Sales", x1, y1)
Dim Dataset2 As New TimeSimpleDataset("Expenses", x1, y2)
```

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);

// 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)

'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)
```

Simple ElapsedTime Dataset

Class ElapsedTimeSimpleDataset

ChartObj
|
+--ChartDataset
|
+--SimpleDataset
|
+-ElapsedTimeSimpleDataset

The **ElapsedTimeSimpleDataset** class uses **TimeSpan** values as one set of the x- or y-values, and floating point values as the other. **TimeSpan** values are actually stored internally as their equivalent millisecond values.

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

ElapsedTimeSimpleDataset constructors

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

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

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

public ElapsedTimeSimpleDataset(
   string sname,
   double[] x
   TimeSpan [] y,
);
```

sname	Specifies the name of the dataset.
x	An array that specifies the x-values (either *doubles* or **TimeSpan** objects) of a dataset.

y	An array that specifies the y-values of a dataset. (either *doubles* or **TimeSpan** objects). The length of the y array must match the length of the x array.

Either x- or y-values should be **TimeSpan** based. 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 an elapsed 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 ElapsedTimeSimpleDataset(
   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.

The only supported format for elapsed time values in a CSV file is d.hh.mm.ss.fff, (3.14:23:12.333 as an example of an elapsed time of three days, 14 hours, 23 minutes, 12 seconds and 333 milliseconds).

You can also modify a point at a time using **SetElapsedTimeXDataValue** (or **SetElapsedTimeYDataValue**) if you are using **TimeSpan** objects**,** and **SetYDataValue**

or **SetXDataValue**) if you use millisecond values. If you need to add new points to the 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.

.

Example of creating a simple elapsed time datasets, extracted from the NewDemosRev2.ElapsedTimeChart example program.

[C#]

```
int  numPoints = 30;
TimeSpan[] x1 = new TimeSpan[numPoints];
double []y1 = new double[numPoints];
double []y2 = new double[numPoints];
int i;
for (i=0; i < numPoints; i++)
{
    x1[i] =  TimeSpan.FromMilliseconds( i * 30 * 1000);  // 30000 milliseconds
increment
 // Or you can use seconds, and the FromSeconds method
//  x1[i] = TimeSpan.FromSeconds(i * 30);  // 30 seconds increment
     if (Math.Sin(x1[i].TotalSeconds / 20.0) > 0)
       y1[i] = 20.0 + 50.0 * (0.5 - ChartSupport.GetRandomDouble()) * (Math.Sin(-
x1[i].TotalSeconds / 5));
     else
      y1[i] = 20.0 + 5.0 * (0.5 - ChartSupport.GetRandomDouble()) * (Math.Sin(-
x1[i].TotalSeconds / 2));

   y2[i] = y1[i] + ((5 + 0.2 * x1[i].TotalSeconds) * (0.5 -
ChartSupport.GetRandomDouble()));
}
ElapsedTimeSimpleDataset Dataset1 = new ElapsedTimeSimpleDataset("First", x1, y1);
ElapsedTimeSimpleDataset Dataset2 = new ElapsedTimeSimpleDataset("Second", x1,
y2);
```

[Visual Basic]

```
Dim numPoints As Integer = 30
Dim x1 As TimeSpan() = New TimeSpan(numPoints - 1) {}
Dim y1 As Double() = New Double(numPoints - 1) {}
Dim y2 As Double() = New Double(numPoints - 1) {}
Dim i As Integer

For i = 0 To numPoints - 1
```

```
   x1(i) = TimeSpan.FromMilliseconds(i * 30 * 1000)
   ' 30000 milliseconds increment
   ' Or you can use seconds, and the FromSeconds method
   ' x1[i] = TimeSpan.FromSeconds(i * 30); // 30 seconds increment
   If Math.Sin(x1(i).TotalSeconds / 20.0R) > 0 Then
      y1(i) = 20.0R + 50.0R * (0.5 - ChartSupport.GetRandomDouble()) * (Math.Sin(-
x1(i).TotalSeconds / 5))
   Else
      y1(i) = 20.0R + 5.0R * (0.5 - ChartSupport.GetRandomDouble()) * (Math.Sin(-
x1(i).TotalSeconds / 2))
   End If

   y2(i) = y1(i) + ((5 + 0.2 * x1(i).TotalSeconds) * (0.5 -
ChartSupport.GetRandomDouble()))
 Next

Dim Dataset1 As New ElapsedTimeSimpleDataset("First", x1, y1)
Dim Dataset2 As New ElapsedTimeSimpleDataset("Second", x1, y2)
```

Contour Plot Dataset

Class ContourDataset

ChartObj
|
+--ChartDataset
|
+--SimpleDataset
|
+-ContourDataset

The **ContourDataset** adds a third dimension (z-values) to the x- and y- values of the simple dataset. It is use exclusively with the contour plotting class, **ContourPlot**.

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.

ContourDataset constructors

```
[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 alorithm 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.
columns	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.
npoints	Specifies the number of xyz data point triplets in the grid array.
x	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.
y	An array, size [npoints] of double that specifies the y-values of the dataset.
z	An array, size [npoints] of double that specifies the z-values of the dataset.
x1, y1, x2, y2	The **SurfaceFunction** sf is evaluated for the range x1,y1 to x2, y2.
sf	The dataset data points are created by evaluating the **SurfaceFunction** across the range x1,y1 to x2, y2.
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 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 tempx, tempy;
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(tempx, tempy, 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
|
+--ChartDataset
|
+--GroupDataset

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

sname	Specifies the name of the dataset.
x	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.
y	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.

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
);
```

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

[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 Dataset1 As GroupDataset = New GroupDataset("First", x1, y1)
```

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")
```

Date/Time Group Dataset

Class TimeGroupDataset

ChartObj
|
+--ChartDataset
|
+--GroupDataset
|
+-TimeGroupDataset

The **TimeGroupDataset** 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(,) _
)

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

sname	Specifies the name of the dataset.
x	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.
y	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.

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
);
```

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 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 **SetDateTimeFomatString** method to change the default date time format. See the documentation for the .Net CF **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 CF 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);
}
TimeGroupDataset Dataset1 = new
       TimeGroupDataset("Stock Data",xValues,stockPriceData);
TimeGroupDataset Dataset1 = new
       TimeGroupDataset("Stock Data",xValues,stockPriceData);
```

[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
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 Dataset1 As New TimeGroupDataset("Stock Data", xValues, stockPriceData)
```

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);
// 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)

'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)
```

Elapsed Time Dataset

Class ElapsedTimeGroupDataset

ChartObj
|
+--ChartDataset
|
+--GroupDataset
|
+--ElapsedTimeGroupDataset

The **ElapsedTimeGroupDataset** 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**.

ElapsedTimeGroupDataset constructors

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

Overloads Public Sub New( _
   ByVal sname As String, _
```

```
   ByVal x As  Double (), _
   ByVal y As Double(,) _
)
[C#]
public ElapsedTimeGroupDataset(
   string sname,
   TimeSpan[] x,
   double[,] y
);


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

sname	Specifies the name of the dataset.
x	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.
y	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.

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 ElapsedTimeGroupDataset(
   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 reading the first value from the row.

The only supported format for elapsed time values in a CSV file is d.hh.mm.ss.fff, (3.14:23:12.333 as an example of an elapsed time of three days, 14 hours, 23 minutes, 12 seconds and 333 milliseconds).

You can also modify a point at a time using **SetElapsedTimeXDataValue** (or **SetElapsedTimeYDataValue**) if you are using **TimeSpan** objects**,** and **SetYDataValue** or **SetXDataValue**) if you use millisecond values. If you need to add new points to the 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.

Example of creating a group datasets from numeric arrays

[C#]

```
int nNumPnts = 5, nNumGroups = 4;
TimeSpan[] xValues = new TimeSpan[nNumPnts];
double[,] groupBarData = new double[nNumGroups, nNumPnts];

xValues[0] = TimeSpan.FromMinutes(1);
groupBarData[0, 0] = 6.3; groupBarData[1, 0] = 3.1;
groupBarData[2, 0] = 2.2; groupBarData[3, 0] = 1.8;

xValues[1] = TimeSpan.FromMinutes(2);
groupBarData[0, 1] = 5.8; groupBarData[1, 1] = 4.3;
groupBarData[2, 1] = 2.8; groupBarData[3, 1] = 1.5;

xValues[2] = TimeSpan.FromMinutes(3);
```

```
groupBarData[0, 2] = 5.5; groupBarData[1, 2] = 4.5;
groupBarData[2, 2] = 2.5; groupBarData[3, 2] = 2.1;

xValues[3] = TimeSpan.FromMinutes(4);
groupBarData[0, 3] = 4.1; groupBarData[1, 3] = 5.4;
groupBarData[2, 3] = 4.1; groupBarData[3, 3] = 3.2;

xValues[4] = TimeSpan.FromMinutes(5);
groupBarData[0, 4] = 3.8; groupBarData[1, 4] = 5.6;
groupBarData[2, 4] = 4.3; groupBarData[3, 4] = 3.3;

ElapsedTimeGroupDataset Dataset1 =
    new ElapsedTimeGroupDataset("ElapsedTimeGroupData", xValues, groupBarData);
```

[VB]

```
Dim nNumPnts As Integer = 5, nNumGroups As Integer = 4
Dim xValues As TimeSpan() = New TimeSpan(nNumPnts - 1) {}
Dim groupBarData As Double(,) = New Double(nNumGroups - 1, nNumPnts - 1) {}

xValues(0) = TimeSpan.FromMinutes(1)
groupBarData(0, 0) = 6.3
groupBarData(1, 0) = 3.1
groupBarData(2, 0) = 2.2
groupBarData(3, 0) = 1.8

xValues(1) = TimeSpan.FromMinutes(2)
groupBarData(0, 1) = 5.8
groupBarData(1, 1) = 4.3
groupBarData(2, 1) = 2.8
groupBarData(3, 1) = 1.5

xValues(2) = TimeSpan.FromMinutes(3)
groupBarData(0, 2) = 5.5
groupBarData(1, 2) = 4.5
groupBarData(2, 2) = 2.5
groupBarData(3, 2) = 2.1

xValues(3) = TimeSpan.FromMinutes(4)
groupBarData(0, 3) = 4.1
groupBarData(1, 3) = 5.4
groupBarData(2, 3) = 4.1
groupBarData(3, 3) = 3.2
```

```
xValues(4) = TimeSpan.FromMinutes(5)

groupBarData(0, 4) = 3.8

groupBarData(1, 4) = 5.6

groupBarData(2, 4) = 4.3

groupBarData(3, 4) = 3.3


Dim Dataset1 As New ElapsedTimeGroupDataset("ElapsedTimeGroupData", xValues,
groupBarData)
```

4. Scaling and Coordinate Systems

ChartScale
- **LinearScale**
- **LogScale**
- **TimeScale**
- **ElapsedTimeScale**

UserCoordinates
- **WorldCoordinates**
  - **WorkingCoordinates**
    - **PhysicalCoordinates**
      - **CartesianCoordinates**
        - **PolarCoordinates**
        - **AntennaCoordinates**
      - **TimeCoordinates**
      - **ElapsedTimeCoordinates**

The starting point for all drawing in a window is the .Net CF 2D device coordinate system. The coordinate system uses a default device resolution of the underlying .Net CF window, regardless of the output device. A .Net CF 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.

Plot area and graph area

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

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. Important parts of the axis, the tick marks and tick mark labels, are usually outside of the plot area. The tick marks and tick mark labels must align perfectly with the coordinate system inside the plot area.

There are many different techniques to align the coordinate system inside the plot area with the coordinate system used in drawing chart objects outside of the plot area. One technique is to maintain the physical coordinate system inside the plot area, and use a normalized coordinate system for the graph area. Whenever a chart object in the graph area, a y-axis tick mark for example, needs to be aligned with the coordinate system inside the plot area, 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 fall on exact pixel boundaries, converting from plot area coordinates to graph area 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 is extended in all four directions – left, right, top and bottom. It is extended so that the physical coordinates of the four corners of the plot area remain unchanged. The four corners of the graph area are assigned calculated, physical coordinate values so that when it is overlaid on to the plot area there is an exact 1:1 correspondence for all points inside the plot area. Once this calculation is made, there is no need to use the physical coordinate system assigned to the plot area. Instead, the physical coordinate system of the graph area 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 while axes objects are not clipped.

For example, assume a graph area with the dimensions of 400x400 units, and a plot area with the dimensions 200x200 centered inside the graph area. This implies that there is a 100 unit boundary around all four sides of the plot area. The desired chart uses a physical coordinate system of (0, 0,100,100). These coordinates apply to the plot area. Instead of using the plot area coordinate system, the coordinate system (-50,-50,150,150) is calculated and used to scale the graph area. This does not guarantee that any point plotted in plot area 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 coordinate system, using only the graph area coordinate system instead. The calculated physical coordinate system applied to the graph area is referred to as the *working* coordinate system.

Coordinate Systems

A **QCChart2D CF** for the **.Net Compact Framework** library uses other coordinate systems mapped onto the default .Net CF device coordinate system. These other coordinate systems include world coordinates, working coordinates, and physical coordinates.

User Coordinates

The **UserCoordinates** class manages a simple viewport drawing system using the .Net CF 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 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) . The world coordinate system applies to the entire graph window, and not just the plot area.

Working Coordinates

The **WorkingCoordinates** class manages a working coordinate system that maps the physical coordinate system of the plot area into a linear, world coordinate system applied to the whole viewport. For example, if the desired chart plot area uses a physical coordinate system of (0.0, 0.0, 100.0, 100.0) and the plot area is centered in the graph area with the plot area ½ the width and height of the graph area, then the coordinate system (-50, -50, 150, 150) is calculated and used to scale the graph area. 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 coordinate system (whether it is linear, logarithmic, date/time, polar, antenna, continuous or discontinuous) into a continuous linear coordinate system. It uses the **WorkingCoordinates** class to map this plot area coordinate system to the entire viewport. The **PhysicalCoordinates** system uses independent scale objects, derived from **ChartScale**, to manage coordinate conversions for the x- and y-dimensions. This way the x-coordinate can use one coordinate conversion object (**LinearScale**, **LogScale**, **TimeScale**) and the y-coordinate another.

There are five concrete implementations of the **PhysicalCoordinates** class: **CartesianCoordinates**, **TimeCoordinates**, **ElapsedTimeCoordinates**, **PolarCoordinates** and **AntennaCoordinates**. 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/date scale for the x-coordinate and a linear or logarithmic scale for the y-coordinate. Use the **ElapsedTimeCoordinates** class when you want a elapsed time scale (no date information) for the x-coordinate and a linear or logarithmic scale for the y-coordinate. Use the **PolarCoordinates** for polar coordinates where the magnitude coordinate is linear and the polar angle coordinate extends from 0 to 360 degrees (or 0 to 2*pi radians) counter-clockwise, starting at 3:00. Use the **AntennaCoordinates** for antenna coordinates where the radius value is linear and the angle coordinate extends from 0 to 360 degrees clockwise, starting at 12:00.

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) is applied to either the graph area, or the plot area of the chart. ***Graph normalized coordinates*** maps the upper left corner of the graph window to the xy coordinates (0.0,0.0) and the lower right corner of the graph area to the xy coordinate (1.0,1.0). ***Plot normalized coordinates*** maps the upper left corner of the plot area to the xy coordinates (0.0,0.0) and the lower right corner of the plot area to the xy coordinates (1.0,1.0).

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 CF. A range of +-10^-30 is just as valid as a range of +-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 +-10^30 range) in a chart scaled for an extremely small range (the +-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 CF floating point format. These types of numbers are often found in databases and representing non-initialized or improperly initialized data. .Net CF 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 in Graph Area

The **WorkingCoordinates** class has a group of methods - **SetGraphBorderFrame**, **SetGraphBorderDiagonal**, and **SetGraphBorderInsets** - that position the plot area 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 in the graph view port is at x = 0.2, y= 0.2, width = 0.6, height = 0.6, specified using graph normalized coordinates.

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

SetGraphBorder methods

```
[Visual Basic]
Overloads Public Sub SetGraphBorderFrame( _
   ByVal border As Rectangle2D _
)
[C#]
public void SetGraphBorderFrame(
   Rectangle2D border
);
```

border Specifies the rectangle defining the plot area border.

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

```
[Visual Basic]
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 width,
```

```
   double height
);
```

where

rLeft	The left x-position of the plot area inside the graph area specified using graph normalized coordinates.
rTop	The top y-position of the plot area inside the graph area specified using graph normalized coordinates.
width	The width of the plot area inside the graph area specified using graph normalized coordinates.
height	The height of the plot area inside the graph area specified using graph normalized coordinates.

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

```
[Visual Basic]
Public Sub SetGraphBorderDiagonal( _
   ByVal rLeft As Double, _
   ByVal rTop As Double, _
   ByVal rRight As Double, _
   ByVal rBottom As Double _
)
[C#]
public void SetGraphBorderDiagonal(
   double rLeft,
   double rTop,
   double rRight,
   double rBottom
);
```

rLeft	The left x-position of the plot area inside the graph area specified using graph normalized coordinates.
rTop	The top y-position of the plot area inside the graph area specified using graph normalized coordinates.

rRight	The right x-position of the plot area inside the graph area specified using graph normalized coordinates.
rBottom	The bottom y-position of the plot area inside the graph area specified using graph normalized coordinates.

This method initializes the insets of the plot area inside the graph area, 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 _
)
[C#]
public void SetGraphBorderInsets(
   double rLeft,
   double rTop,
   double rRight,
   double rBottom
);
```

rLeft	The left inset of the plot area inside the graph area specified using graph normalized coordinates.
rTop	The top inset of the plot area inside the graph area specified using graph normalized coordinates.
rRight	The right inset of the plot area inside the graph area specified using graph normalized coordinates.
rBottom	The bottom inset of the plot area inside the graph area specified using graph normalized coordinates.

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

[C#]

```
CartesianCoordinates simpleScale;
```

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

// Use ONE of the example below
// Example #1
simpleScale.SetGraphBorderFrame(new Rectangle2D(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 Rectangle2D(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)
```

Linear and Logarithmic Coordinate Scaling

Class CartesianCoordinates

PhysicalCoordinates

|

+-- CartesianCoordinates

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

There are three main ways to scale the plot area:

- 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 both x and y. If you already know the range for x and y for the plot area, you can scale the plot area explicitly. The example below uses a **CartesianCoordinates** constructor to initialize the coordinates to the proper values.

CartesianCoordinates constructor with explicit scaling

[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)
```

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

[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)
```

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};

SimpleDataset dataset = new SimpleDataset("Sales", xData, yData);
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 dataset As SimpleDataset = New SimpleDataset("Sales", xData, yData)
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};

SimpleDataset dataset = new SimpleDataset("Sales", xData, yData);
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 dataset As SimpleDataset = New SimpleDataset("Sales", xData, yData)
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};

SimpleDataset dataset1 = new SimpleDataset("Sales1",xData1,yData1);
SimpleDataset dataset2 = new SimpleDataset("Sales2",xData2,yData2);
SimpleDataset dataset3 = new SimpleDataset("Sales3",xData3,yData3);
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 dataset1 As SimpleDataset = New SimpleDataset("Sales1", xData1, yData1)
Dim dataset2 As SimpleDataset = New SimpleDataset("Sales2", xData2, yData2)
Dim dataset3 As SimpleDataset = New SimpleDataset("Sales3", xData3, yData3)
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 =
          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 **QCChart2D CF** for the **.Net Compact Framework** 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 **QCChart2D CF** for the **.Net Compact Framework** 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 31minor 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 **QCChart2D CF for the .Net Compact Framework** 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 **QCChart2D CF** for the **.Net Compact Framework** 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 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- and y-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 and linear for the y-scale scale. The y-scale 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 and y for the plot area, you can scale the plot area 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;

TimeCoordinates simpleTimeScale;
simpleTimeScale = new TimeCoordinates(xMin, yMin, xMax, 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 =  105

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

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;
TimeCoordinates simpleTimeScale = new TimeCoordinates();

simpleTimeScale.SetTimeCoordinateBounds (xMin, yMin, xMax, 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 =  105

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

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

[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};

TimeSimpleDataset dataset =  new TimeSimpleDataset("Sales", xData, yData);
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 dataset As TimeSimpleDataset = New TimeSimpleDataset("Sales", xData, yData)
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- 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#]

```
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};

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

[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 dataset As TimeSimpleDataset = New TimeSimpleDataset("Sales", xData, yData)
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};
// 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);
TimeSimpleDataset dataset2 = new TimeSimpleDataset("Sales2",xData,yData2);
TimeSimpleDataset dataset3 = new TimeSimpleDataset("Sales3",xData,yData3);
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}
' 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)
Dim dataset2 As TimeSimpleDataset = New TimeSimpleDataset("Sales2",xData,yData2)
Dim dataset3 As TimeSimpleDataset = New TimeSimpleDataset("Sales3",xData,yData3)
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 _
)
```

```
[C#]
public TimeCoordinates(
   ChartCalendar dstart,
   double y1,
   ChartCalendar dstop,
   double y2,
   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 _
)
```

```
[C#]
public void SetTimeCoordinateBounds(
   ChartCalendar dstart,
   double y1,
   ChartCalendar dstop,
   double y2,
   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 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
);
```

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; // msecs cooresponding to 9:30 AM
long stoptod = 16 * 60 * 60 * 1000; // msecs 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 ' msecs cooresponding to 9:30 AM
Dim stoptod As Long = 16 * 60 * 60 * 1000 ' msecs 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

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 _
)
[C#]
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; // msecs cooresponding to 9:30 AM
long stoptod = 16 * 60 * 60 * 1000; // msecs 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 ' msecs cooresponding to 9:30 AM
Dim stoptod As Long = 16 * 60 * 60 * 1000 '  msecs 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; // msecs cooresponding to 9:30 AM
long stopTime = 16 * 60 * 60 * 1000; // msecs cooresponding to 4:00 PM

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

Class ElapsedTimeCoordinates

PhysicalCoordinates

|

+-- CartesianCoordinates

|

+-- ElapsedTimeCoordinates

The **ElapsedTimeCoordinates** class scales the chart plot area for a physical coordinate system which uses an elapsed time scale in combination with a linear or logarithmic scaling. The elapsed time scale uses milliseconds as the time base, so all time values should be represented using their milliseconds equivalent value, i.e. a value of 30 seconds is represented by the value 30000. The axis labeling class, **ElapsedTimeAxisLabels**, converts the millisecond values to equivalent seconds values, i.e., the value 30000 milliseconds will be displayed as 30 seconds in the format "00:00:30".

There are three main ways to scale the plot area:

- Scale the minimum and maximum x- and y- values explicitly. You can scale the the elapse time axis using **TimeSpan** objects, or using millisecond values, i.e. an elapsed time range of 0 – 30 seconds would be scaled for 0-30000.
- Use an auto-scale method to 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.

ElapsedTime Coordinate Scaling

The default coordinate system for the **ElapsedTimeCoordinates** class is elapsed time for the x-dimension and linear for the y dimension. If you already know the range for x and y for the plot area, you can scale the plot area explicitly.

The example below uses a **ElapsedTimeCoordinates** constructor to initialize the coordinates to the proper values.

Scale for elapsed time using the ElapsedTimeCoordinates constructor with explicit scaling for a range of 30 seconds.

[C#]

```
double xMin = 0; // starting elapsed time is 0
double xMax = 30 * 1000; // ending elpase time is 30 seconds
double yMin = 0;
double yMax =  105;

ElapsedTimeCoordinates simpleScale;
simpleScale = new ElapsedTimeCoordinates(xMin, yMin, xMax, yMax);

TimeSpan xTSMin = TimeSpan.FromSeconds(0); // starting elapsed time is 0
TimeSpan xTSMax = TimeSpan.FromSeconds(30); // ending elpase time is 30 seconds
simpleScale = new ElapsedTimeCoordinates(xTSMin, yMin, xTSMax, yMax);
```

[Visual Basic]

```
Dim xMin As Double = 0  ' starting elapsed time is 0
Dim xMax As Double= 30 * 1000' ending elpase time is 30 seconds
Dim yMin As Double = 0
Dim yMax As Double =  105

Dim simpleScale As ElapsedTimeCoordinates
simpleScale = New ElapsedTimeCoordinates (xMin, yMin, xMax, yMax)
// or scale using TimeSpan values

Dim xTSMin  As TimeSpan = TimeSpan.FromSeconds(0) ' starting time is 0
Dim xTSMax  As TimeSpan = TimeSpan.FromSeconds(30) 'ending time is 30 seconds
simpleScale = new ElapsedTimeCoordinates(xTSMin, yMin, xTSMax, yMax)
```

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

Example of explicit scaling of a ElapsedTimeCoordinates object using the ElapsedTimeCoordinates.SetCoordinateBounds method

[C#]

```
Dim xMin As Double = 0  ' starting elapsed time is 0
Dim xMax As Double= 30 * 1000 ' ending elpase time is 30 seconds
double yMin = 0;
double yMax =  105;
ElapsedTimeCoordinates simpleScale = new ElapsedTimeCoordinates ();

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

[Visual Basic]

```
Dim xMin As Double = 0  ' starting elapsed time is 0
Dim xMax As Double= 30 * 1000 ' ending elpase time is 30 seconds
Dim yMin As Double = 0
Dim yMax As Double =  105

Dim simpleScale As ElapsedTimeCoordinates = New ElapsedTimeCoordinates ()
simpleScale.SetCoordinateBounds(xMin, yMin, xMax, yMax)
```

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 ElapsedTimeCoordinates object using a single dataset

[C#]

```
double [] xData = {1000,2000,3000,4000,5000,6000,7000,8000,9000,10000};
double [] yData = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9};

ElapsedTimeSimpleDataset dataset = new ElapsedTimeSimpleDataset ("Sales", xData,
yData);
ElapsedTimeCoordinates simpleScale  = new ElapsedTimeCoordinates ();
simpleScale.AutoScale(dataset);
```

[Visual Basic]

```
Dim xData() As Double  = {1000,2000,3000,4000,5000,6000,7000,8000,9000,10000}
Dim yData() As Double  = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9}
```

```
Dim dataset As ElapsedTimeSimpleDataset = New ElapsedTimeSimpleDataset ("Sales",
xData, yData)

Dim simpleScale As ElapsedTimeCoordinates = New ElapsedTimeCoordinates ()

simpleScale.AutoScale(dataset)
```

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

Example of auto-scaling a ElapsedTimeCoordinates 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 = {1000,2000,3000,4000,5000,6000,7000,8000,9000};

double [] yData = {11, 22, 33, 44, 55, 46, 33, 25, 14};


ElapsedTimeSimpleDataset dataset = new ElapsedTimeSimpleDataset ("Sales", xData,
yData);

ElapsedTimeCoordinates simpleScale  = new ElapsedTimeCoordinates ();

simpleScale.AutoScale(dataset);

simpleScale.SetScaleStartY(0);

simpleScale.SetScaleStopY(100.0);
```

[Visual Basic]

```
Dim xData() As Double  = {1000,2000,3000,4000,5000,6000,7000,8000,9000}

Dim yData() As Double  = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9}
```

```
Dim dataset As ElapsedTimeSimpleDataset = New ElapsedTimeSimpleDataset ("Sales",
xData, yData)

Dim simpleScale As ElapsedTimeCoordinates = new ElapsedTimeCoordinates ()

simpleScale.AutoScale(dataset)

simpleScale.SetScaleStartY(0)

simpleScale.SetScaleStopY(100.0)
```

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

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

[C#]

```
double [] xData1 = {1000,2000,3000,4000,5000,6000,7000,8000,9000,10000};

double [] yData1 = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9};

double [] xData2 = 1000,2000,3000,4000,5000,6000,7000,8000,9000,10000};

double [] yData2 = {20, 12, 43, 54, 15, 26, 63, 25, 24, 19};

double [] xData3 = 1000,2000,3000,4000,5000,6000,7000,8000,9000,10000};

double [] yData3 = {30, 52, 13, 64, 25, 76, 13, 35, 24, 19};


ElapsedTimeSimpleDataset dataset1 = new ElapsedTimeSimpleDataset
("Sales1",xData1,yData1);

ElapsedTimeSimpleDataset dataset2 = new ElapsedTimeSimpleDataset
("Sales2",xData2,yData2);

ElapsedTimeSimpleDataset dataset3 = new ElapsedTimeSimpleDataset
("Sales3",xData3,yData3);

ElapsedTimeSimpleDataset [] datasetsArray = new ElapsedTimeSimpleDataset [3];

datasetsArray[0] = dataset1;

datasetsArray[1] = dataset2;

datasetsArray[2] = dataset3;

ElapsedTimeCoordinates simpleScale  = new ElapsedTimeCoordinates ();

simpleScale.AutoScale(datasetsArray);
```

[Visual Basic]

```
Dim xData1() As Double = {1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000,
10000}

Dim yData1() As Double = {10, 22, 33, 44, 55, 46, 33, 25, 14, 9}

Dim xData2() As Double = {1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000,
10000}

Dim yData2() As Double = {20, 12, 43, 54, 15, 26, 63, 25, 24, 19}
```

```
Dim xData3() As Double = {1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000,
10000}
Dim yData3() As Double = {30, 52, 13, 64, 25, 76, 13, 35, 24, 19}


Dim dataset1 As ElapsedTimeSimpleDataset = New ElapsedTimeSimpleDataset ("Sales1",
xData1, yData1)
Dim dataset2 As ElapsedTimeSimpleDataset = New ElapsedTimeSimpleDataset ("Sales2",
xData2, yData2)
Dim dataset3 As ElapsedTimeSimpleDataset = New ElapsedTimeSimpleDataset ("Sales3",
xData3, yData3)
Dim datasetsArray(2) As ElapsedTimeSimpleDataset
datasetsArray(0) = dataset1
datasetsArray(1) = dataset2
datasetsArray(2) = dataset3
Dim simpleScale As ElapsedTimeCoordinates = New ElapsedTimeCoordinates ()
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)
```

Polar Coordinate Systems

Class PolarCoordinates

PhysicalCoordinates
|
+-- CartesianCoordinates
|
+-- PolarCoordinates

The magnitude and the polar angle of a point define its position in a chart scaled for polar coordinates. The magnitude can have any value greater than 0.0 and the polar angle any positive or negative value. A polar angle range of 0 to 2 pi radians (0 to 360 degrees) sweeps a complete circle in polar coordinates.

A polar coordinate system uses a Cartesian coordinate system scaled for plus/minus the polar magnitude. The following equations convert from polar coordinates to Cartesian coordinates.

x = magnitude * cos (angle)

y = magnitude * sine (angle)

magnitude	Polar coordinate magnitude
angle	Plot coordinate angle
x	Cartesian x-coordinate
y	Cartesian y-coordinate

The **PolarCoordinates** class is an extension of the **CartesianCoordinates** class and it automatically handles these conversions. The only important parameter that needed for the creation of a **PolarCoordinates** object is the polar magnitude, since the polar angle always has a range 0 to 2 pi radians (0 to 360 degrees).

PolarCoordinates contructors

The first way to create a **PolarCoordinates** object is to use the constructor that specifies the polar magnitude directly.

[C#]

```
double polarmagnitude = 5.0;
PolarCoordinates polarscale = new PolarCoordinates(polarmagnitude);
```

[Visual Basic]

```
Dim polarmagnitude As Double = 5.0
Dim polarscale AS PolarCoordinates = New PolarCoordinates(polarmagnitude)
```

Or you can use an auto-scale routine to analyze a dataset and select the appropriate polar magnitude.

[C#]

```
double []angleData = {.20,.60,1.40,1.70,2.50,4.0,5.0, 6.0};   // In Radians
double []magnitudeData = { 20, 33, 44, 55, 46, 33, 54, 64};
SimpleDataset dataset = new SimpleDataset("Control", angleData, magnitudeData);
PolarCoordinates pPolarTransform = new PolarCoordinates();
pPolarTransform.AutoScale(dataset, ChartObj.AUTOAXES_FAR);
```

[Visual Basic]

```
Dim angleData() As Double = {0.2, 0.6, 1.4, 1.7, 2.5, 4.0, 5.0, 6.0} ' In Radians
Dim magnitudeData() As Double = {20, 33, 44, 55, 46, 33, 54, 64}
Dim dataset As SimpleDataset = _
          New SimpleDataset("Control", angleData, magnitudeData)
Dim pPolarTransform As PolarCoordinates = New PolarCoordinates()
pPolarTransform.AutoScale(dataset, ChartObj.AUTOAXES_FAR)
```

Antenna Coordinate Systems

Class AntennaCoordinates

PhysicalCoordinates

|

+-- CartesianCoordinates

|

+-- AntennaCoordinates

An antenna coordinate's point is defined by its radial and angular values. The radial and angle values can be positive or negative. An angle range of 0 to 360 degrees clockwise, starting at 12:00, sweeps a complete circle in antenna coordinates.

AntennaCoordinates are defined by specifying a starting and ending value for the radius. Unlike a polar chart, these values can be positive or negative. Antenna coordinates always have an angular range 0 to 360 degrees.

AntennaCoordinates constructors

The first way to create a **AntennaCoordinates** object is to use the constructor that specifies the minimum and maximum values of the radius: **AntennaCoordinates(minvalue, maxvalue).**

[C#]

```
double minvalue = -40;
double maxvalue = 20;

AntennaCoordinates antennacoords = new AntennaCoordinates (minvalue, maxvalue);
```

[Visual Basic]

```
Dim minvalue As Double = -40
Dim maxvalue As Double = 20

Dim antennacoords As AntennaCoordinates = new AntennaCoordinates (minvalue,
maxvalue)
```

Or you can use an auto-scale routine to analyze a dataset and select the appropriate antenna radius limits.

[C#]

```
double []angleData = {0, 30, 60, 90, 120, 150, 180};   // In degrees
double []radiusData = { -35, -31, -5, 12, 14, -14, -30};
SimpleDataset dataset = new SimpleDataset("Control", angleData, radiusData);
AntennaCoordinates antennacoords = new AntennaCoordinates ();
antennacoords.AutoScale(dataset, ChartObj.AUTOAXES_FAR);
```

[Visual Basic]

```
Dim angleData() As Double = {0, 30, 60, 90, 120, 150, 180} ' In degrees
Dim radiusData () As Double = {-35, -31, -5, 12, 14, -14, -30}
Dim dataset As SimpleDataset = _
          New SimpleDataset("Control", angleData, radiusData)
Dim antennacoords As AntennaCoordinates = New AntennaCoordinates ()
antennacoords.AutoScale(dataset, 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** or **InvertScaleY** 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};

SimpleDataset dataset = new SimpleDataset("Sales", xData, yData);
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 dataset As SimpleDataset = New SimpleDataset("Sales", xData, yData)
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 CF 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 CF component that can be placed and positioned in **UserControl** windows using a .Net CF layout managers.

Control
|
+-- ChartView

The **ChartView** class has several constructors that position the ChartView class within a parent window.

ChartView constructor

```
[Visual Basic]

Overloads Public Sub New()

Overloads Public Sub New( _
   ByVal positionRect As Rectangle, _
   ByVal parent As Control _
)

Overloads Public Sub New( _
   ByVal positionRect As Rectangle, _
   ByVal parent As Form _
)

Overloads Public Sub New( _
   ByVal positionRect As Rectangle, _
   ByVal parent As Panel _
)

Overloads Public Sub New( _
   ByVal positionRect As Rectangle, _
   ByVal parent As TabPage _
)


[C#]

public ChartView()

public ChartView( Rectangle positionRect, System.Windows.Forms.Form parent)

public ChartView( Rectangle positionRect, System.Windows.Forms.UserControl
parent)
```

```
public ChartView( Rectangle positionRect, System.Windows.Forms.Panel parent)
public ChartView( Rectangle positionRect, System.Windows.Forms.TabPage parent)
```

positionRect	A rectangle that specifies the size and position of the ChartView in the parent window.
parent	The parent window of the ChartView. This parent window can take the form of System.Windows.Forms Form, Control, Panel or TabPage class.

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.

There are two princible methods of adding a **ChartView** object to a .Net CF Form. The first is to add the **ChartView** control to the program at design time. This is the mode most programmers are familiar with since they are used to populating an empty form with butons, text boxes, etc. in order to build their displays. The second method is to add the **ChartView** to the form under program control, bypassing the design mode instantiation of the control completely. The second method is useful if you need to customize the size and position of the **ChartView** control based on the resolution of the target device.

Using ChartView with the Visual Studio Design Mode

Display the Form where you want the **ChartView** control using the Visual Studio design mode. Select the **ChartView** control from the Toolbox and drop it on the form. Resize the control to the size you want. If you size the control larger than the viewable area of the underlying form, horizontal and vertical scrollbars will automatically appear so that you can pan the form left/right and up/down. Most of our example programs size the **ChartView** larger than the underlying form.

This assumes that you have explicity added the **ChartView** control to the Visual Studio Toolbox. To add the **ChartView** control to the VS Toolbox, right click on the Toolbox and select **Choose Items**. From the Choose Toolbox Items dialog, use the **Browse** button and go to the Quinn-Curtis\DotNet\Lib folder and select QCChart2DNetCF.DLL. OK out of the Choose Toolbox Items dialog. You will now see a **ChartView** control in the tool box list of controls.

When you add a **ChartView** control to a Form, it places the default instantiation code in the form's Form.Designer.cs module. Starting with VS 2005, Microsoft has decided to physically separate the designer generated code in a Form.Designer module, while

keeping your user written code in a Form.cs module. The default name used for the first instance of the **ChartView** control added to a program is chartView1. The variable chartView1 becomes global to the entire Form class. You will see a reference to chartView1 in all of our example programs. You will find the definition of the chartView1 variable in the form's Designer.cs module. In our example programs, in the form's user code area (Form.cs or Form.vb) we typically assign chartView1 to a local variable named chartVu, and then customize the graph using the chartVu variable name. You will also need to make sure that you reference com.quinncurtis.chart2dnetcf in the using (or Imports for VB) section of any module that references **ChartView**. See the example program UserChartExample1 for a simple example.

ChartView example (extracted from the example program UserChartExample1, Form1)

[C#]

```
ChartView chartVu = chartView1;


Background background = new Background(pTransform1, ChartObj.PLOT_BACKGROUND,
Color.White);
chartVu.AddChartObject(background);


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


LinearAxis yAxis = new LinearAxis(pTransform1, ChartObj.Y_AXIS);
chartVu.AddChartObject(yAxis);
```

[VB]

```
Dim chartVu As ChartView = ChartView1


Dim background As New Background(pTransform1, ChartObj.PLOT_BACKGROUND,
Color.White)
chartVu.AddChartObject(background)


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


Dim yAxis As New LinearAxis(pTransform1, ChartObj.Y_AXIS)
        chartVu.AddChartObject(yAxis)
```

Once you place an instance of a **ChartView** object on a form at design time, don't exepect to start "programming" the control using the control's designe time property

table. A design time property table is poor way of creating a chart with a custom look and we do not implement properties that allow you to define a chart at design time. Instead you define the chart in the body of the form that contains the chart. See any of our examples.

Adding a ChartView control at runtime

The **ChartView** class can also be instantiated at runtime, without placing it on a form at design time. You can use the default **ChartView** constructor, or one of the special constructors that simplify adding an instance of the control to a form at runtime. See the list of **ChartView** constructors a couple of pages back. In general you pass the **ChartView** constructor a parent object (Form, Panel or TabbedControl) and a positioning rectangle that defines the position and size of the **ChartView** object. In order to add a **ChartView** control at runtime, the **ChartView** control does NOT need to be added to the Visual Studio Toolbox. You will need to add the QCChart2DNetCF.DLL to the project by right clicking project name in the Solution Explorer and selecting **Add Reference**. Browse to the QCChart2DNetCF.DLL in the Quinn-Curtis\DotNet\lib folder and select it. You will also need to make sure that you reference com.quinncurtis.chart2dnetcf in the using (or Imports for VB) section of any module that references **ChartView**.

ChartView constructor example (extracted from the example program UserChartExample2, Form1)

[C#]

```
public Form1()
{
   InitializeComponent();
   InstantiateChartView();
   InitializeChart();
}


public void InstantiateChartView()
{
   // Allow room for a menu in the form
   Rectangle graphRect = new Rectangle(1, 1,this.ClientRectangle.Width - 2,
this.ClientRectangle.Height - 2);


   // Instantiate chart control
  chartVu = new ChartView(graphRect, this);
  // Set preferred size of graph, forces rescaling of fonts
  // for smaller displayes
  chartVu.PreferredSize  = chartView1.ClientSize;
}
```

```
public void InitializeChart()
{
  Font theFont;
  int numPoints = 25;
  double[] x1 = new double[numPoints];
  double[] y1 = new double[numPoints];
  double[] y2 = new double[numPoints];
  double[] y3 = new double[numPoints];
  int i;

.
.
.
}
```

In VB, the constructor for the Form is in Form.Designer.vb module and not the user programmer Form.vb module. For that reason the chart creation code is placed in the Form_Load event.

[Visual Basic]

```
Imports com.quinncurtis.chart2dnetcf


Public Class Form1
    Dim chartView1 As ChartView = Nothing


    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        InstantiateChartView()
        If (chartView1 IsNot Nothing) Then
            InitializeChart()
        End If
    End Sub


    Public Sub InstantiateChartView()
        ' Allow room for a menu in the form
        Dim graphRect As Rectangle = New Rectangle(1, 1, Me.ClientRectangle.Width
- 2, Me.ClientRectangle.Height - 2)


        ' Instantiate chart control
        chartView1 = New ChartView(graphRect, Me)
    End Sub
```

```
    Public Sub InitializeChart()
        Dim chartVu As ChartView = chartView1
        .
        .
        .
    End Sub 'InitializeChart

End Class
```

Adding a UserControl derived from ChartView at runtime

You can derive your own chart class from **ChartView**, and use that control instead of the base **ChartView** class. Right click on the project in the Solution Explorer and select **Add | User Control**. Choose the **UserControl** template. This will add a basic template for a user defined UserControl1 to your project. Go to the UserControl1.cs (or UserControl1.vb) and change the inheritance from **UserControl** to **ChartView**.

In order to add a **ChartView** control at runtime, the **ChartView** control does NOT need to be added to the Visual Studio Toolbox. You will need to add the QCChart2DNetCF.DLL to the project by right clicking project name in the Solution Explorer and selecting **Add Reference**. Browse to the QCChart2DNetCF.DLL in the Quinn-Curtis\DotNet\lib folder and select it. You will also need to make sure that you reference com.quinncurtis.chart2dnetcf in the using (or Imports for VB) section of any module that references **ChartView**.

ChartView constructor example (extracted from the example program UserChartExample3, Form1)

[C#]

```
public partial class Form1 : Form
{
  // SEE THE UserControl1.cs module for details of UserControl1
  UserControl1 chartView1 = null;
  public Form1()
  {
    InitializeComponent();
    InstantiateChartView();
    if (chartView1 != null)
      chartView1.InitializeChart();
  }
  public void InstantiateChartView()
```

```
  {
    // Allow room for a menu in the form
    Rectangle graphRect = new Rectangle(1, 1, this.ClientRectangle.Width - 2,
this.ClientRectangle.Height - 2);


            // Instantiate chart control
     chartView1 = new UserControl1();
     chartView1.Location = graphRect.Location;
     chartView1.Size = graphRect.Size;
     chartView1.PreferredSize = graphRect.Size;
     this.Controls.Add(chartView1);
  }
}
```

In VB, the constructor for the Form is in Form.Designer.vb module and not the user programmer Form.vb module. For that reason the chart creation code is place in the Form_Load event.

[Visual Basic]

```
Public Class Form1
  ' SEE THE UserControl1.vb module for details of UserControl1
    Dim ChartView1 As UserControl1 = Nothing


    Public Sub InstantiateChartView()
        ' Allow room for a menu in the form
        Dim graphRect As Rectangle = New Rectangle(1, 1, Me.ClientRectangle.Width
- 2, Me.ClientRectangle.Height - 2)


        ' Instantiate chart control and add it to the form
        ChartView1 = New UserControl1()
        ChartView1.Location = graphRect.Location
        ChartView1.Size = graphRect.Size
        ChartView1.PreferredSize = graphRect.Size
        Me.Controls.Add(ChartView1)
    End Sub


    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        InstantiateChartView()
        ChartView1.InitializeChart()
    End Sub
End Class
```

Note how the **ChartView.PreferredSize** property is used to establish the base size for font scaling. If the final chart window is larger than the PreferredSize, the fonts are scaled larger, if the final chart window is smaller than the PreferredSize, the fonts are scaled smaller.

ChartView.AddChartObject example (extracted from the example program BigChartDemo, class LabeledDatapoints)

[C#]

```
ChartView chartVu = chartView1;
SimpleDataset Dataset1 = new SimpleDataset("First",x1,y1);
CartesianCoordinates  pTransform1 =
    new CartesianCoordinates( ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
pTransform1.SetGraphBorderDiagonal(0.15, .15, .90, 0.8) ;
xAxis = new LinearAxis(pTransform1, ChartObj.X_AXIS);
chartVu.AddChartObject(xAxis);
```

[Visual Basic]

```
Dim chartVu As ChartView = ChartView1
Dim Dataset1 As New SimpleDataset("First", x1, y1)
Dim pTransform1 As New CartesianCoordinates( ChartObj.LINEAR_SCALE, _
               ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 0.7)
Dim xAxis As New LinearAxis(pTransform1, ChartObj.X_AXIS)
chartVu.AddChartObject(xAxis)
```

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 ordered 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 background has a z-value of 10 and a graph area background has a z-value of 9, forcing graph area 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.
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**, **ContourPlot** and **PolarPlot** 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  = chartView1;


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

```
xAxis.SetZOrder(30);
chartVu.AddChartObject(xAxis);
```

[Visual Basic]

```
Dim chartVu As ChartView = ChartView1

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  = chartView1;

LinearAxis xAxis = new LinearAxis(pTransform1, ChartObj.X_AXIS);
chartVu.AddChartObject(xAxis);
.
.
.
xAxis.SetColor(Color.Red);
chartVu.UpdateDraw();
```

[Visual Basic]

```
Dim chartVu As ChartView = ChartView1

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 CF container object such as a **Control**. A .Net CF 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** or **PolarCoordinates**) for each chart, and to position the plot area 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 BigChartDemo, class OHLCFinPlot)

[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)
```

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 areas 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 in the chart view is set using one of the coordinate systems **SetGraphBorder**… methods. Many of the examples use this technique, including OHLFinPlot, MultiAxes, LinearAxes, LogAxes and DateAxes1 and 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. 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, 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. If the charts aspect ratio changes drastically, the font size 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.

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 Point2D, _
   ByVal classname As String _
) As GraphObj

[C#]
public GraphObj FindObj(
   Point2D testpoint,
   string classname
);
```

Parameters

If the graph has multiple overlapping objects of the same type, you can return the nth object intersecting the test point.

```
[Visual Basic]
Overloads Public Function FindObj( _
   ByVal testpoint As Point2D, _
   ByVal classname As String, _
   ByVal nthhit As Integer _
) As GraphObj

[C#]
public GraphObj FindObj(
   Point2D testpoint,
   string classname,
   int nthhit
);
```

testpoint	The current position of the mouse in .Net CF device coordinates.
classname	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.
nthhit	Specifies to return the nth 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. Background Colors and Gradients

Background

Two rectangular areas act as a backdrop for the other graphical elements of a chart. The first is the rectangle formed by the **ChartView** class. This rectangle is the *graph area* and all elements of the chart (axes, labels, plots, titles, etc.) are within its bounds. The second area is the plot area. That area is within the graph area. Its position within the graph area 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. Other chart objects (axes, axes labels, titles, etc.) need to reside outside of the plot area and these objects clip to the graph area. The plot area can have a background different from that of the graph background. Often a contrast between the graph area background and the plot area background produces a more visually pleasing chart.

Class Background

GraphObj
|
+-- Background

The **Background** class paints the graph area background or the plot area background. One instance of the class can only paint one area, either the graph area or the plot area. If you want unique fill properties for both, you need to create two instances of the class. The **Background** class uses one of the following techniques to fill the background:

- solid color
- simple color gradient defined using two RGB colors

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
);
```

transform	The coordinate system associated with the chart background. The *transform* defines where the plot area fits in the graph area.
bgtype	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 of the chart, while specifying the GRAPH_BACKGROUND type fills the entire graph area of the chart.
startcolor	Specifies the starting color value of the gradient.
stopcolor	Specifies the ending color value of the gradient.
dir	Specifies the direction of the gradient.

Should 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.

The example below defines a simple linear gradient for the graph background area (extracted from the example program BigChartDemo, class LineFill)

[C#]

```
pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(DatasetArray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
pTransform1.SetGraphBorderDiagonal(0.15, .1, .92, 0.75) ;
Background background = new Background( pTransform1, ChartObj.GRAPH_BACKGROUND,
        Color.FromArgb(100,50,255), Color.FromArgb(40,25,120), ChartObj.Y_AXIS);
chartVu.AddChartObject(background);
```

[Visual Basic]

```
Dim pTransform1 As TimeCoordinates = New TimeCoordinates()
pTransform1.AutoScale(DatasetArray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

pTransform1.SetGraphBorderDiagonal(0.15, 0.1, 0.92, 0.75)
Dim background As New Background(pTransform1, _
        ChartObj.GRAPH_BACKGROUND, Color.FromArgb(100, 50, 255), _
        Color.FromArgb(40, 25, 120), ChartObj.Y_AXIS)
chartVu.AddChartObject(background)
```

7. Axes

Axis

- **LinearAxis**
  - **ElapsedTimeAxes**
  - **PolarAxes**
  - **AntennaAxes**
- **LogAxis**
- **TimeAxis**

Chart axes describe for the viewer the physical coordinate system used to scale the plot area 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
- Axes for linear,date/time, elapsed time, logarithmic, polar and antenna, 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 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.

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 four concrete axis types supported by the **QCChart2D CF** for the **.Net Compact Framework** library:

Axis Type	Class
Linear	**LinearAxis**
Logarithmic	**LogAxis**
Date/time	**TimeAxis**
ElapsedTime	**ElapsedTimeAxis**
Polar	**PolarAxes**
Antenna	**AntennaAxes**

The four 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
);
```

rgbcolor	Sets the primary line color for the chart object.
linewidth	Sets the line width, in device coordinates, for the chart object.
linestyle	Sets the line style for the chart object.

Linear Axes

Class LinearAxis

GraphObj
|
+--Axis
|
+-- LinearAxis

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, time, and polar 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 Intercept

An axis resides in a 2-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 coordinate needed to position the axis is the axis intercept. The axis intercept specifies the y-coordinate position for the x-axis, and the x-coordinate position for the y-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), or a y-axis (Y_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 method

```
[Visual Basic]
Public Sub SetAxisIntercept( _
   ByVal intercept As Double _
)
```

```
[C#]
public void SetAxisIntercept(
   double intercept
);
```

SetAxisTicks method

```
 [Visual Basic]
Overloads Public Sub SetAxisTicks( _
   ByVal tickorigin As Double, _
   ByVal tickspace As Double, _
   ByVal nticksperma jor As Integer _
)
```

```
[Visual Basic]
Overloads Public Sub SetAxisTicks( _
   ByVal tickorigin As Double, _
   ByVal tickspace As Double, _
   ByVal nminorticksperma jor As Integer, _
   ByVal minorticklength As Double, _
   ByVal majorticklength As Double, _
   ByVal tickdir As Integer _
)
```

```
[C#]
public void SetAxisTicks(
   double tickorigin,
   double tickspace,
   int nticksperma jor
);
```

```
public void SetAxisTicks(
   double tickorigin,
   double tickspace,
   int nminorticksperma jor,
   double minorticklength,
   double majorticklength,
   int tickdir
);
```

intercept	Sets the intercept of this axis with the perpendicular axis in physical coordinates.
tickorigin	The tick marks start at this value.
tickspace	Specifies the spacing between minor tick marks.
nticksperma jor	Specifies the number of minor tick marks per major tick mark.
minorticklength	The length of minor tick marks, in .Net CF device coordinates.
majorticklength	The length of major tick marks, in .Net CF device coordinates.
tickdir	The direction of the tick marks. Use one of the tick mark direction constants: AXIS_MIN, AXIS_CENTER, or AXIS_MAX.

Use the **SetColor** method 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   = chartView1;

// 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 = ChartView1

' 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 xAxisOrigin = 0.0;
double xAxisMinorTickSpace = 1.0;
int xAxisMinorTicksPerMajor = 5;
double xAxisMinorTickLength = 5;
double xAxisMajorTickLength = 10;
int xAxisTickDirection = ChartObj.AXIS_MIN;

xAxis.SetAxisIntercept(xAxisIntercept);
xAxis.SetAxisTicks(xAxisOrigin, xAxisMinorTickSpace,
                   xAxisMinorTicksPerMajor, xAxisMinorTickLength,
                   xAxisMajorTickLength, xAxisTickDirection);
```

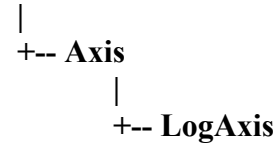
[Visual Basic]

```
Dim xAxisIntercept As Double = -5
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.SetAxisTicks(xAxisOrigin, xAxisMinorTickSpace, _
             xAxisMinorTicksPerMajor, xAxisMinorTickLength, _
              xAxisMajorTickLength, xAxisTickDirection)
```

Logarithmic Axes

Scientific, engineering and financial applications often require the use of logarithmic axe. 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 an intercept value, the same as a linear 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), or a y-axis (Y_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 method

```
[Visual Basic]
Public Sub SetAxisIntercept( _
   ByVal intercept As Double _
)

[C#]
public void SetAxisIntercept(
   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 CF device coordinates.

majorticklength The length of major tick marks, in .Net CF device coordinates.

ticdir The direction of the tick marks. Use one of the tick mark direction constants: AXIS_MIN, AXIS_CENTER, or AXIS_MAX.

The **SetColor** method is 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   = chartView1;

// 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 = ChartView1

' 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;
// 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.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.

Note – The **TimeAxis** class is **not** used to create an axis to display elapsed time. Use a **ElapsedTimeCoordinates** system in combination with a **ElapsedTimeAxis** to create an elapsed time axis. It is the **ElapsedTimeAxisLabels** that give the elapsed time axis its time axis look, i.e. 10:30:22.

Class TimeAxis

GraphObj
|
+--Axis
|
+-- TimeAxis

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.

Date/Time Axis Intercept

A date/time axis has an intercept value, the same as a linear axis. Since the intercept value is specified using the scale of the perpendicular axis, if the perpendicular axis is linear, 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.

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
);
```

transform	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 *transform* time-scale.
dstart	The starting date value for the axis.
dstop	The ending date value for the axis
ntickmarkbase	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 method

```
[Visual Basic]
Public Sub SetAxisIntercept( _
   ByVal intercept As Double _
)

[C#]
public void SetAxisIntercept(
   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.
minorticklength	Specifies the length of a minor tick mark in .Net CF device coordinates.
majorticklength	Specifies the length of a major tick mark in .Net CF device coordinates.
tickdir	Specifies the direction of the tick marks with respect to axis line. Use one of the following tick direction constants: AXIS_MIN, AXIS_CENTER, AXIS_MAX.

Customize the line and tick mark drawing properties of the axis using the **SetColor** method.

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 = ChartView1

' 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 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.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   = chartView1;

// 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 = ChartView1

' Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis)
chartVu.AddChartObject(yAxis)
```

Elapsed Time Axes

Class ElapsedTimeAxis

GraphObj
|
+--Axis
|
+-- LinearAxis
|
+-- ElapsedTimeAxes

The **ElapsedTimeAxis** is subclassed from the **LinearAxis** class and has much in common with it. The only difference between the two is the way in which major and minor tick marks are calculated in the **CalcAutoAxis** method. The **ElapsedTimeAxis** takes into account the base 60 of seconds per minute and minutes per hour, and the base 24 of hours per day. Read the sections:

- **Linear Axis Minimum and Maximum**
- **Linear Minor and Major Tick Mark Intervals**
- **Linear Axis Intercept**
- **Linear Axis Tick Mark Origin**

under the discussion of **LinearAxis**.

Creating a Elapsed Time Axis

There are two main constructors for **ElapsedTimeAxes** objects. The first **ElapsedTimeAxes** constructor assumes that the axis extents match the extents of the underlying coordinate system, *transform*. The second **ElapsedTimeAxes** constructor sets the axis extents to the specified minimum and maximum values, regardless of the underlying coordinate system.

ElapsedTimeAxes 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 TimeSpan, _
   ByVal maxval As TimeSpan _
)



[C#]
public ElapsedTimeAxis(
   PhysicalCoordinates transform,
   int axtype
);



public ElapsedTimeAxis (
   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), or a y-axis (Y_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 method

```
[Visual Basic]
Public Sub SetAxisIntercept( _
   ByVal intercept As Double _
)

[C#]
public void SetAxisIntercept(
```

```
   double intercept
);
```

SetAxisTicks method

```
 [Visual Basic]
Overloads Public Sub SetAxisTicks( _
   ByVal tickorigin As Double, _
   ByVal tickspace As Double, _
   ByVal nticksperma jor As Integer _
)
```

```
[Visual Basic]
Overloads Public Sub SetAxisTicks( _
   ByVal tickorigin As Double, _
   ByVal tickspace As Double, _
   ByVal nminorticksperma jor As Integer, _
   ByVal minorticklength As Double, _
   ByVal majorticklength As Double, _
   ByVal tickdir As Integer _
)
```

```
[C#]
public void SetAxisTicks(
   double tickorigin,
   double tickspace,
   int nticksperma jor
);
```

```
public void SetAxisTicks(
   double tickorigin,
   double tickspace,
   int nminorticksperma jor,
   double minorticklength,
   double majorticklength,
   int tickdir
);
```

intercept	Sets the intercept of this axis with the perpendicular axis in physical coordinates.
tickorigin	The tick marks start at this value.
tickspace	Specifies the spacing between minor tick marks.
nticksperma jor	Specifies the number of minor tick marks per major tick mark.
minorticklength	The length of minor tick marks, in .Net device coordinates.
majorticklength	The length of major tick marks, in .Net device coordinates.
tickdir	The direction of the tick marks. Use one of the tick mark direction constants: AXIS_MIN, AXIS_CENTER, or AXIS_MAX.

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 elapsed time axis example

[C#]

```
// Define the coordinate system
TimeSpan xMin = TimeSpan.FromSeconds(0);
TimeSpan xMax = TimeSpan.FromSeconds(15);
double yMin = 0;
double yMax =  105;
ElapsedTimeCoordinates simpleScale =
               new ElapsedTimeCoordinates (xMin, yMin, xMax, yMax);

// Create the x- and y-axes
ElapsedTimeAxis xAxis = new ElapsedTimeAxis (simpleScale, ChartObj.X_AXIS);
ElapsedTimeAxis yAxis = new ElapsedTimeAxis (simpleScale, ChartObj.Y_AXIS);

// 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 TimeSpan = TimeSpan.FromSeconds(0)
Dim xMax As TimeSpan = TimeSpan.FromSeconds(15)
Dim yMin As Double = 0
Dim yMax As Double =  15
Dim simpleScale As ElapsedTimeCoordinates =  _
               New ElapsedTimeCoordinates (xMin, yMin, xMax, yMax)

' Create the x- and y-axes
Dim xAxis As ElapsedTimeAxis = New ElapsedTimeAxis (simpleScale, ChartObj.X_AXIS)
Dim yAxis As ElapsedTimeAxis = New ElapsedTimeAxis (simpleScale, ChartObj.Y_AXIS)

' Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis)
chartVu.AddChartObject(yAxis)
```

Polar Axes

Polar axes provide the visual scale needed to compare data values that use polar coordinates. A polar axis consists of two parts. The first part is a pair of linear x- and y-axes intersecting at the center, Cartesian coordinate (0, 0). The second part of a polar axis is a circle with radius R, centered on the origin.

Class PolarAxes

GraphObj

```
        |
        +--Axis
                |
                +-- LinearAxis
                        |
                        +-- PolarAxes
```

The **PolarAxes** class creates a polar axes object that combines linear x- and y-axes for measurement of the polar magnitude, and a circular axis for measurement of the polar angle. The **PolarAxes** class extends the **LinearAxis** class. This is useful because the **LinearAxis** already has member variables that define properties and draw the tick marks for the circular axis. The **PolarAxes** class also includes uses two additional **LinearAxis** objects in support of the x and y linear axes used in the drawing of the polar magnitude axes.

Polar Axis Minimum and Maximum

Polar axes have only one scaling value, the maximum value of the polar magnitude, designated R. The minimum value is always 0.0. The limits of the x- and y-axis are set to +-R with the intercept for each axis set to 0.0. The polar angle scale is always 360 degrees, corresponding to a full circle.

Polar Axis Minor and Major Tick Mark Intervals

Polar axes use two sets of tick mark properties; one set for the x- and y-axes and the other set for the circular axis. The x- and y-axes use the same values for the major and minor tick mark spacing, partitioning the axes between +-R endpoints. The circular axis also uses major and minor tick marks, analogous to the hour and minute marks of a clock.

Creating polar axes

There is only one constructor for **PolarAxes** objects.

```
PolarAxes(PolarCoordinates transform)
```

transform The *transform* coordinate system defines the axis extents of the polar axes.

The **PolarAxes** constructor assumes that the axis extents match the extents of the underlying coordinate system, *transform*.

Other axis properties: minor tick mark spacing, number of minor tick marks per major tick mark, tick mark direction and tick mark lengths are automatically calculated using an auto-axis method. These properties can be explicitly set if you need to override the automatically calculated values.

```
[Visual Basic]
Overloads Public Sub SetPolarAxesTicks( _
   ByVal axestickspace As Double, _
   ByVal axesnticksperma jor As Integer, _
   ByVal angletickspace As Double, _
   ByVal anglenticksperma jor As Integer _
)

Overloads Public Sub SetPolarAxesTicks( _
   ByVal axestickspace As Double, _
   ByVal axesnticksperma jor As Integer, _
   ByVal angletickspace As Double, _
   ByVal anglenticksperma jor As Integer, _
   ByVal minorticklength As Double, _
   ByVal majorticklength As Double, _
   ByVal tickdir As Integer _
)
```

```
[C#]
public void SetPolarAxesTicks(
   double axestickspace,
   int axesnticksperma jor,
   double angletickspace,
   int anglenticksperma jor
);

public void SetPolarAxesTicks(
   double axestickspace,
   int axesnticksperma jor,
   double angletickspace,
   int anglenticksperma jor,
   double minorticklength,
   double majorticklength,
   int tickdir
);
```

axestickspace	Specifies the spacing between minor tick marks for the x- and y-axes.
axesntickspermajor	Specifies the number of minor tick marks per major tick mark for the x- and y-axes.
angletickspace	Specifies the spacing, in degrees, between minor tick marks for the radial axis.
anglentickspermajor	Specifies the number of minor tick marks per major tick mark for the radial axis.
minorticlength	The length of minor tick marks, in .Net CF device coordinates.
majorticlength	The length of major tick marks, in .Net CF device coordinates.
tickdir	The direction of the tick marks. Use one of the tick mark direction constants: AXIS_MIN, AXIS_CENTER, or AXIS_MAX.

Use the **SetColor** method to customize the drawing properties of the lines used to draw the axes lines and tick marks.

Simple polar axes example

[C#]

```
double polarmagnitude = 5.0;
PolarCoordinates polarscale = new PolarCoordinates(polarmagnitude);
PolarAxes polarAxes = new PolarAxes(polarscale);
// Create the ChartView object to place graph objects in.
ChartView chartVu   = chartView1;
// Add the polar axes to the chartVu object
chartVu.AddChartObject(polarAxes);
```

[Visual Basic]

```
Dim polarmagnitude As Double = 5.0
Dim polarscale As PolarCoordinates = New PolarCoordinates(polarmagnitude)
Dim polarAxes As PolarAxes = New PolarAxes(polarscale)
' Create the ChartView object to place graph objects in.
Dim chartVu As ChartView = ChartView1
' Add the polar axes to the chartVu object
chartVu.AddChartObject(polarAxes)
```

Custom polar axes example

[C#]

```
double polarmagnitude = 15.0;
PolarCoordinates polarscale = new PolarCoordinates(polarmagnitude);
PolarAxes polarAxes = new PolarAxes(polarscale);

double axestickspace = 1;
int axesntickspermajor = 5;
double angletickspace = 50;
int anglentickspermajor = 6;
double minorticlength = 5;
double majorticlength = 10;
int tickdir = ChartObj.AXIS_CENTER;

polarAxes.SetPolarAxesTicks(axestickspace, axesntickspermajor,
               angletickspace, anglentickspermajor,
               minorticlength, majorticlength,
               tickdir);
// Create the ChartView object to place graph objects in.
ChartView chartVu   = chartView1;
// Add the polar axes to the chartVu object
chartVu.AddChartObject(polarAxes);
```

[Visual Basic]

```
 Dim polarmagnitude As Double = 15.0
 Dim polarscale As PolarCoordinates = New PolarCoordinates(polarmagnitude)
 Dim polarAxes As PolarAxes = New PolarAxes(polarscale)
 Dim axestickspace As Double = 1
 Dim axesntickspermajor As Integer
 Dim angletickspace As Double = 50
 Dim anglentickspermajor As Integer
 Dim minorticlength As Double = 5
 Dim majorticlength As Double = 10
 Dim tickdir As Integer = ChartObj.AXIS_CENTER

 polarAxes.SetPolarAxesTicks(axestickspace, axesntickspermajor, _
             angletickspace, anglentickspermajor, _
```

```
            minorticlength, majorticlength, _
            tickdir)
' Create the ChartView object to place graph objects in.
Dim chartVu As ChartView = ChartView1
' Add the polar axes to the chartVu object
chartVu.AddChartObject(polarAxes)
```

Antenna Axes

Antenna axes provide the visual scale needed to compare data values that use antenna coordinates. An antenna axis consists of two parts. The first part is a linear y-axis extending from the origin to the outer edge of the radial scale. The second part of an antenna axis is a circle with a radius equal to the range of the radial scale, centered on the origin.

Class AntennaAxes

GraphObj
|
+--Axis
|
+-- LinearAxis
|
+-- AntennaAxes

The **AntennaAxes** class creates an antenna axes object that combines a linear y-axis for measurement of the radial values, and a circular axis for the measurement of the angular values. The **AntennaAxes** class extends the **LinearAxis** class. This is useful because the **LinearAxis** already has member variables that define properties and draw the tick marks for the circular axis.

Antenna Axis Minimum and Maximum

Antenna axes use two scaling values, a minimum and maximum radius value. The radius minimum value is set at the origin of the coordinate system, and the radius maximum value at the outer edge. The radius starting and ending values can be positive or negative. The maximum value should always be greater than the minimum value though. The angular scale is always 360 degrees, corresponding to a full circle. The angular scale starts with 0 degrees at 12:00 and increases clockwise.

Antenna Axis Minor and Major Tick Mark Intervals

Antenna axes use two sets of tick mark properties; one set for the y-axis and the other set for the circular axis. The y-axis has major and minor tick mark properties, as does the circular axis.

Creating antenna axes

There is only one constructor for **AntennaAxes** objects.

```
AntennaAxes(AntennaCoordinates transform)
```

transform The *transform* coordinate system defines the axis extents of the antenna axes.

The **AntennaAxes** constructor assumes that the axis extents match the extents of the underlying coordinate system, *transform*.

Other axis properties: minor tick mark spacing, number of minor tick marks per major tick mark, tick mark direction and tick mark lengths are automatically calculated using an auto-axis method. These properties can be explicitly set if you need to override the automatically calculated values.

```
[Visual Basic]
Overloads Public Sub SetAntennaAxesTicks( _
   ByVal axestickspace As Double, _
   ByVal axesntickspermajor As Integer, _
   ByVal angletickspace As Double, _
   ByVal anglentickspermajor As Integer _
)

Overloads Public Sub SetAntennaAxesTicks( _
   ByVal axestickspace As Double, _
   ByVal axesntickspermajor As Integer, _
   ByVal angletickspace As Double, _
   ByVal anglentickspermajor As Integer, _
   ByVal minorticklength As Double, _
   ByVal majorticklength As Double, _
   ByVal tickdir As Integer _
)

[C#]
public void SetAntennaAxesTicks(
   double axestickspace,
   int axesntickspermajor,
   double angletickspace,
   int anglentickspermajor
);

public void SetAntennaAxesTicks(
   double axestickspace,
   int axesntickspermajor,
```

```
   double angletickspace,
   int anglentickspermajor,
   double minorticklength,
   double majorticklength,
   int tickdir
);
```

axestickspace	Specifies the spacing between minor tick marks for the x- and y-axes.
axesntickspermajor	Specifies the number of minor tick marks per major tick mark for the x- and y-axes.
angletickspace	Specifies the spacing, in degrees, between minor tick marks for the radial axis.
anglentickspermajor	Specifies the number of minor tick marks per major tick mark for the radial axis.
minorticlength	The length of minor tick marks, in .Net device coordinates.
majorticlength	The length of major tick marks, in .Net device coordinates.
tickdir	The direction of the tick marks. Use one of the tick mark direction constants: AXIS_MIN, AXIS_CENTER, or AXIS_MAX.

Use the **SetLineWidth**, **SetLineStyle** and **SetColor** methods to customize the drawing properties of the lines used to draw the axes lines and tick marks.

Simple antenna axes example

[C#]

```
double minvalue  = -40, maxvalue = 20;
AntennaCoordinates antennascale = new AntennaCoordinates(minvalue, maxvalue);
AntennaAxes antennaAxes = new AntennaAxes(antennascale);
// Add the Antenna axes to the chartVu object
chartVu.AddChartObject(antennaAxes);
```

[Visual Basic]

```
Dim minvalue As Double = -40
Dim maxvalue As Double = 20
Dim antennascale As AntennaCoordinates = _
     New AntennaCoordinates(minvalue, maxvalue)
```

```
Dim antennaAxes As AntennaAxes = New AntennaAxes(antennascale)
' Add the Antenna axes to the chartVu object
chartVu.AddChartObject(antennaAxes)
```

Custom antenna axes example

[C#]

```
double minvalue  = -40, maxvalue = 20;
AntennaCoordinates antennascale = new AntennaCoordinates(minvalue, maxvalue);
AntennaAxes antennaAxes = new AntennaAxes(antennascale);

double axestickspace = 1;
int axesntickspermajor = 5;
double angletickspace = 5;
int anglentickspermajor = 6;
double minorticlength = 5;
double majorticlength = 10;
int tickdir = ChartObj.AXIS_CENTER;

antennaAxes.SetAntennaAxesTicks(axestickspace, axesntickspermajor,
               angletickspace, anglentickspermajor,
               minorticlength, majorticlength,
               tickdir);
// Add the Antenna axes to the chartVu object
chartVu.AddChartObject(antennaAxes);
```

[Visual Basic]

```
Dim minvalue As Double = -40
Dim maxvalue As Double = 20
 Dim antennascale As AntennaCoordinates = _
       New AntennaCoordinates(minvalue, maxvalue)
 Dim antennaAxes As AntennaAxes = New AntennaAxes(antennascale)
 Dim axestickspace As Double = 1
 Dim axesntickspermajor As Integer = 5
 Dim angletickspace As Double = 5
 Dim anglentickspermajor As Integer = 6
 Dim minorticlength As Double = 5
 Dim majorticlength As Double = 10
 Dim tickdir As Integer = ChartObj.AXIS_CENTER
```

```
 antennaAxes.SetAntennaAxesTicks(axestickspace, axesntickspermajor, _
             angletickspace, anglenticksperpermajor, _
             minorticlength, majorticlength, _
             tickdir)
' Add the Antenna axes to the chartVu object
chartVu.AddChartObject(antennaAxes)
```

8. Axis Labels

AxisLabels
- **NumericAxisLabels**
- **TimeAxisLabels**
- **ElapsedTimeAxisLabels**
- **StringAxisLabels**
- **PolarAxesLabels**
- **AntennaAxesLabels**

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 six axis labels classes: the **AxisLabels** abstract base class and concrete subclasses **NumericAxisLabels**, **TimeAxisLabels, ElapsedTimeAxisLabels, StringAxisLabels**, **PolarAxesLabels** and **AntennaAxesLabels.**.

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 ($1.5x10^{-5}$)
- 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)

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/yyy hh:mm:ss)

ElapsedTimeAxisLabels
The **ElapsedTimeAxisLabels** are used in combination with the **ElapsedTimeAxis** type**.**

lapsed time formats (hh:mm:ss, hh:mm, mm:ss, mm:ss.fff, etc.)

PolarAxesLabels
This class displays numeric labels exclusively for the **PolarAxes** class. It uses labels in the same format as the **NumericAxisLabels**.

AntennaAxesLabels
This class displays numeric labels exclusively for the **AntennaAxes** class. It uses labels in the same format as the **NumericAxisLabels**.

Class AxisLabels

GraphObj
|
+-- ChartText
|
+-- AxisLabels

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 place 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

|

+-- ChartText

|

+-- AxisLabels

|

+-- NumericAxisLabels

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.
CURRENCEYBUSINESSFORMAT	A ‘$’ character is appended to the front of the label and the values are abbreviated the same as the BUSINESSFORMAT
CURRENCEYFORMAT	A ‘$’ 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
);
```

font	The font object used to display the axis label text.
rotation	The rotation, in degrees, of label text in the normal viewing plane.
labdir	The justification of the axis label (AXIS_MIN or AXIS_MAX) with respect to the tick mark endpoint.
decimal	Sets the number of digits to the right of the decimal point for numeric axis labels.
labelends	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
labcolor	The color of the label text.
format	Sets the numeric format for the axis labels. Use one of the numeric format constants: DECIMALFORMAT, SCIENTIFICFORMAT, EXPONENTFORMAT, BUSINESSFORMAT, ENGINEERINGFORMAT, PERCENTFORMAT, CURRENCEYFORMAT, 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   = chartView1;

// 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 = ChartView1
' 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
|
+-- ChartText
|
+-- AxisLabels
|
+-- TimeAxisLabels

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 **QCChart2D CF** for the **.Net Compact Framework** 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	"MMMMM 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
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, event 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
);
```

font	The font object used to display the axis label text.
rotation	The rotation, in degrees, of label text in the normal viewing plane.
labdir	The justification of the axis label (AXIS_MIN or AXIS_MAX) with respect to the tick mark endpoint.
decimal	Sets the number of digits to the right of the decimal point for numeric axis labels.
labelends	Ignored for time axis labels
labcolor	The color of the label text.
format	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   = chartView1;
```

```
// 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 = ChartView1

 ' 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)
```

Elapsed Time Axis Labels

Class ElapsedTimeAxisLabels

GraphObj
|
+-- ChartText
|
+-- AxisLabels
|
+-- ElapsedTimeAxisLabels

The **ElapsedTimeAxisLabels** class extends the **AxisLabels** class and provides for elapsed time labels. It adds extensive time formatting capability. Use it to label axes created using the **ElapsedTimeAxis** class.

Elapsed Time Label formats

A time axis label can take several forms. The TimeFormat property controls the elapsed time format.

TimeFormat Format Constant	**Example String Result**
TIMEDATEFORMAT_MS	12:33
TIMEDATEFORMAT_24HMS	23:12:33
TIMEDATEFORMAT_24HM	23:12

The TIMEDATEFORMAT_MS and TIMEDATEFORMAT formats can also have a decimal precision, appending a decimal point and the specified number of significant digits to the right of the time lable seconds, i.e. 12:33.7432. This is set using the **AxisLabelsDecimalPos** property.

ElapsedTimeAxis Labels constructor

There is only one main constructor for **TimeAxisLabels** objects.

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal baseaxis As ElapsedTimeAxis _
)

[C#]
public ElapsedTimeAxisLabels(
   ElapsedTimeAxis baseaxis
);
```

baseaxis This is the elapsed time axis the axis labels are for.

Other axis label properties: font, rotation, time 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 timeformat As Integer, _
   ByVal labelends As Integer, _
   ByVal labcolor As Color _
)

[C#]
public void SetAxisLabels(
```

```
   Font font,
   double rotation,
   int labdir,
   int decimalpos,
   int timeformat,
   int labelends,
   Color labcolor
);
```

SetAxisLabelsFormat method

```
[Visual Basic]
Public Sub SetAxisLabelsFormat( _
   ByVal format As Integer _
)
[C#]
public void SetAxisLabelsFormat(
   int format
);
```

font	The font object used to display the axis label text.
rotation	The rotation, in degrees, of label text in the normal viewing plane.
labdir	The justification of the axis label (AXIS_MIN or AXIS_MAX) with respect to the tick mark endpoint.
decimal	Sets the number of digits to the right of the decimal point for elapsed time axis labels.
labelends	Ignored for time axis labels
labcolor	The color of the label text.
timeformat	Sets the numeric format for the axis labels. Use one of the time format constants: TIMEDATEFORMAT_MS, TIMEDATEFORMAT_24HMS, TIMEDATEFORMAT_24HM.

Simple ElapsedTimeAxisLabels example (extracted from the NewDemosRev2.ElapsedTimeChart example program)

[C#]

```
TimeSpan[] x1 = new TimeSpan[numPoints];
```

```
double []y1 = new double[numPoints];
double []y2 = new double[numPoints];
int i;

for (i=0; i < numPoints; i++)
{
  // Initialize Data
  .
  .
  .
}

theFont = new Font("Microsoft Sans Serif", 10,  FontStyle.Bold);
ElapsedTimeSimpleDataset Dataset1 = new ElapsedTimeSimpleDataset("First", x1, y1);
ElapsedTimeSimpleDataset Dataset2 =
    new ElapsedTimeSimpleDataset("Second", x1, y2);


ElapsedTimeCoordinates pTransform1 =
    new ElapsedTimeCoordinates(ChartObj.ELAPSEDTIME_SCALE, ChartObj.LINEAR_SCALE);
pTransform1.ElapsedTimeAutoScale(Dataset2, ChartObj.X_AXIS,
    ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
pTransform1.SetGraphBorderDiagonal(0.15, .15, .90, 0.70) ;
.
.
.
ElapsedTimeAxis xAxis = new ElapsedTimeAxis(pTransform1, ChartObj.X_AXIS);
chartVu.AddChartObject(xAxis);

LinearAxis yAxis = new LinearAxis(pTransform1, ChartObj.Y_AXIS);
chartVu.AddChartObject(yAxis);

ElapsedTimeAxisLabels xAxisLab = new ElapsedTimeAxisLabels(xAxis);
xAxisLab.SetTextFont(theFont);
xAxisLab.AxisLabelsDayFormat = ChartObj.ELAPSEDTIMEFORMAT_NEXTTODAYSTRING;
chartVu.AddChartObject(xAxisLab);

NumericAxisLabels yAxisLab = new NumericAxisLabels(yAxis);
yAxisLab.SetTextFont(theFont);
chartVu.AddChartObject(yAxisLab);
```

[Visual Basic]

```
Dim x1 As TimeSpan() = New TimeSpan(numPoints - 1) {}
Dim y1 As Double() = New Double(numPoints - 1) {}
Dim y2 As Double() = New Double(numPoints - 1) {}
Dim i As Integer

For i = 0 To numPoints - 1
            .
.
.
Next

theFont = New Font("Microsoft Sans Serif", 10, FontStyle.Bold)
Dim Dataset1 As New ElapsedTimeSimpleDataset("First", x1, y1)
Dim Dataset2 As New ElapsedTimeSimpleDataset("Second", x1, y2)

Dim pTransform1 As New ElapsedTimeCoordinates(ChartObj.ELAPSEDTIME_SCALE, _
     ChartObj.LINEAR_SCALE)
pTransform1.ElapsedTimeAutoScale(Dataset2, ChartObj.X_AXIS, _
     ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 0.7)
.
.
.
Dim xAxis As New ElapsedTimeAxis(pTransform1, ChartObj.X_AXIS)
chartVu.AddChartObject(xAxis)

Dim yAxis As New LinearAxis(pTransform1, ChartObj.Y_AXIS)
chartVu.AddChartObject(yAxis)
Dim xAxisLab As New ElapsedTimeAxisLabels(xAxis)
xAxisLab.SetTextFont(theFont)
xAxisLab.AxisLabelsDayFormat = ChartObj.ELAPSEDTIMEFORMAT_NEXTTODAYSTRING
chartVu.AddChartObject(xAxisLab)

Dim yAxisLab As New NumericAxisLabels(yAxis)
yAxisLab.SetTextFont(theFont)
chartVu.AddChartObject(yAxisLab)
```

Polar Axes Labels

Class PolarAxesLabels

GraphObj
|
+-- ChartText
|
+-- AxisLabels
|
+-- NumericAxisLabels
|
+-- PolarAxesLabels

The **PolarAxesLabels** class extends the **NumericAxisLabels** class and creates labels for objects of the **PolarAxes** class. The **PolarAxesLabels** class labels the two parts of the polar axes: the x- and y-axes pair defining the polar magnitude, and the polar angle circle, bounding the x- and y-axes. The class extends the **NumericAxisLabels** class and uses that class's methods and properties for managing the label properties.

The x- and y-axes have extents of +-R. The only labels needed for these axes are for the positive section of the x-axis. The easiest way to manage this is to create a local x-axis that extends from 0 to +R. This local axis is not drawn, but is used to create a **NumericAxisLabels** object for the class. This object draws the labels for the positive section of the x-axis.

PolarAxisLabels contructor

There is only one main constructor for **PolarAxesLabels** objects.

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal baseaxis As PolarAxes _
)

[C#]
public PolarAxesLabels(
   PolarAxes 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 labcolor As Color _
)
[C#]
public void SetAxisLabels(
   Font font,
   Color labcolor
);
```

SetAxisLabelsFormat method

```
[Visual Basic]
Public Sub SetAxisLabelsFormat( _
   ByVal format As Integer _
)
[C#]
public void SetAxisLabelsFormat(
   int format
);
```

font	The font object used to display the axis label text.
labcolor	The color of the label text.
format	Sets the numeric format for the axis labels. Use one of the numeric format constants: DECIMALFORMAT, SCIENTIFICFORMAT, EXPONENTFORMAT, BUSINESSFORMAT, ENGINEERINGFORMAT, PERCENTFORMAT, CURRENCEYFORMAT, CURRENCYBUSINESSFORMAT.

Polar axes labels example

[C#]

```
double polarmagnitude = 5.0;
PolarCoordinates polarscale = new PolarCoordinates(polarmagnitude);
PolarAxes polarAxes = new PolarAxes(polarscale);

PolarAxesLabels polarAxesLabels = new PolarAxesLabels(polarAxes);
polarAxesLabels.SetAxisLabelsFormat(ChartObj.DECIMALFORMAT);
polarAxesLabels.SetAxisLabelsDecimalPos(2);

// Create the ChartView object to place graph objects in.
ChartView chartVu   = chartView1;
```

```
// Add the polar axes to the chartVu object
chartVu.AddChartObject(polarAxes);
chartVu.AddChartObject(polarAxesLabels);
```

[Visual Basic]

```
Dim polarmagnitude As Double = 5.0
Dim polarscale As PolarCoordinates = New PolarCoordinates(polarmagnitude)
Dim polarAxes As PolarAxes = New PolarAxes(polarscale)

Dim polarAxesLabels As PolarAxesLabels = New PolarAxesLabels(polarAxes)
polarAxesLabels.SetAxisLabelsFormat(ChartObj.DECIMALFORMAT)
polarAxesLabels.SetAxisLabelsDecimalPos(2)

' Create the ChartView object to place graph objects in.
Dim chartVu As ChartView = ChartView1
' Add the polar axes to the chartVu object
chartVu.AddChartObject(polarAxes)
chartVu.AddChartObject(polarAxesLabels)
```

Antenna Axes Labels

Class AntennaAxesLabels

GraphObj
|
+-- ChartText
|
+-- AxisLabels
|
+-- NumericAxisLabels
|
+-- AntennaAxesLabels

The **AntennaAxesLabels** class extends the **NumericAxisLabels** class and creates labels for objects of the **AntennaAxes** class. The **AntennaAxesLabels** class labels the two parts of the antenna axes: the y-axis displaying the radius limits, and the angular circle bounding the y-axis. The class extends the **NumericAxisLabels** class and uses that class's methods and properties for managing the label properties.

AntennaAxisLabels constructor

There is only one main constructor for **AntennaAxesLabels** objects.

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal baseaxis As AntennaAxes _
)
[C#]
public AntennaAxesLabels(
   AntennaAxes 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 labcolor As Color _
)
[C#]
public void SetAxisLabels(
   Font font,
   Color labcolor
);
```

SetAxisLabelsFormat method

```
[Visual Basic]
Public Sub SetAxisLabelsFormat( _
   ByVal format As Integer _
)
[C#]
public void SetAxisLabelsFormat(
   int format
);
```

font The font object used to display the axis label text.

labcolor The color of the label text.

format Sets the numeric format for the axis labels. Use one of the numeric format constants: DECIMALFORMAT, SCIENTIFICFORMAT, EXPONENTFORMAT, BUSINESSFORMAT, ENGINEERINGFORMAT, PERCENTFORMAT, CURRENCEYFORMAT, CURRENCYBUSINESSFORMAT.

Antenna axes labels example

[C#]

```
double minvalue  = -40, maxvalue = 20;
AntennaCoordinates antennascale = new AntennaCoordinates(minvalue, maxvalue);
AntennaAxes antennaAxes = new AntennaAxes(antennascale);

chartVu.AddChartObject(antennaAxes);

AntennaAxesLabels antennaAxesLabels = new AntennaAxesLabels (antennaAxes);
antennaAxesLabels.SetAxisLabelsFormat(ChartObj.DECIMALFORMAT);
antennaAxesLabels.SetAxisLabelsDecimalPos(2);

chartVu.AddChartObject(antennaAxesLabels);
```

[Visual Basic]

```
Dim minvalue As Double = -40
Dim maxvalue As Double = 20
Dim antennascale As AntennaCoordinates = _
     New AntennaCoordinates(minvalue, maxvalue)
Dim antennaAxes As AntennaAxes = New AntennaAxes(antennascale)

chartVu.AddChartObject(antennaAxes)

 Dim antennaAxesLabels As AntennaAxesLabels = New AntennaAxesLabels (antennaAxes)
 antennaAxesLabels.SetAxisLabelsFormat(ChartObj.DECIMALFORMAT)
 antennaAxesLabels.SetAxisLabelsDecimalPos(2)

chartVu.AddChartObject(antennaAxesLabels)
```

9. Axis Grids

Grid

PolarAxesGrid
AntennaGrid

Axis grids are solid, dotted or dashed lines, aligned with the axis tick marks and which extend across the plot area 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 both an x- and y-axis. The three axis grid classes are **Grid, PolarGrid** and **AntennaGrid.**

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 both an x- and y-axis. The grid aligns with the tick marks of one axis, and extends across the plot area 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.

Grid constuctor

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal xaxis As Axis, _
   ByVal yaxis As Axis, _
   ByVal gridaxistype As Integer, _
   ByVal gridtype As Integer _
)
[C#]
public Grid(
   Axis xaxis,
   Axis yaxis,
   int gridaxistype,
```

```
   int gridtype
);
```

xaxis	The x-axis associated with the grid.
yaxis	The y-axis associated with the grid.
gridaxistype	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 or Y_AXIS.
gridtype	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
);
```

rgbcolor	Sets the primary line color for the chart object.
linewidth	Sets the line width, in device coordinates, for the chart object.
linestyle	Sets the line style for the chart object.

Grid 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 grids
Grid gridX =  new Grid(xAxis, yAxis, ChartObj.X_AXIS, ChartObj.GRID_MAJOR);
//Change default grid line properties
gridX.SetColor(Color.Gray);

Grid gridY =  new Grid(xAxis, yAxis, ChartObj.Y_AXIS, ChartObj.GRID_MAJOR);
//Change default grid line properties
gridY.SetColor(Color.Black);

// Create the ChartView object to place graph objects in.
ChartView chartVu   = chartView1;

// Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis);
chartVu.AddChartObject(yAxis);
```

```
chartVu.AddChartObject(gridX);
chartVu.AddChartObject(gridY);
```

[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)
' Create the grids
Dim gridX As Grid = New Grid(xAxis, yAxis, ChartObj.X_AXIS, ChartObj.GRID_MAJOR)
' Change default grid line properties
gridX.SetColor(Color.Gray)

Dim gridY As Grid = New Grid(xAxis, yAxis, ChartObj.Y_AXIS, ChartObj.GRID_MAJOR)
' Change default grid line properties
gridY.SetColor(Color.Black)

' Create the ChartView object to place graph objects in.
Dim chartVu As ChartView = ChartView1

' Add the x- and y-axes to the chartVu object
chartVu.AddChartObject(xAxis)
chartVu.AddChartObject(yAxis)
chartVu.AddChartObject(gridX)
chartVu.AddChartObject(gridY)
```

Polar Grids

Class PolarGrid

GraphObj

```
|
+-- Grid
      |
      +-- PolarGrid
```

The **PolarGrid** class defines a grid for polar axes. The polar grid consists of two parts: the magnitude grid and the polar angle grid. The magnitude grid consists of a group of concentric circles centered on the origin and aligned with the x- and y-axis tick marks. The polar angle grid consists of a group of radial lines, aligned with the angle tick marks and starting at the origin extending to the outer polar angle circle.

PolarGrid constructors

There are two **PolarGrid** constructors.

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal polaraxis As PolarAxes, _
   ByVal gridtype As Integer _
)


Overloads Public Sub New( _
   ByVal polaraxis As PolarAxes, _
   ByVal gridmagtype As Integer, _
   ByVal gridangletype As Integer _
)


[C#]
public PolarGrid(
   PolarAxes polaraxis,
   int gridtype
);

public PolarGrid(
   PolarAxes polaraxis,
   int gridmagtype,
   int gridangletype
);
```

polaraxis	The polar axes associated with the grid.
gridtype	Specifies if the magnitude and angular 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 polar axis.
gridmagtype	Specifies if the magnitude 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 polar axis.
gridangletype	Specifies if the angular 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 polar axis.

The **SetColor** method is used to customize the drawing properties of the lines used to draw the axes lines and tick marks.

Polar grid example

[C#]

```
double polarmagnitude = 5.0;
PolarCoordinates polarscale = new PolarCoordinates(polarmagnitude);
PolarAxes polarAxes = new PolarAxes(polarscale);
PolarGrid polarGrid =  new PolarGrid(polarAxes, ChartObj.GRID_MAJOR);
// Create the ChartView object to place graph objects in.
ChartView chartVu   = chartView1;
// Add the polar axes to the chartVu object
chartVu.AddChartObject(polarAxes);
chartVu.AddChartObject(polarGrid);
```

[Visual Basic]

```
Dim polarmagnitude As Double = 5.0
Dim polarscale As PolarCoordinates = New PolarCoordinates(polarmagnitude)
Dim polarAxes As PolarAxes = New PolarAxes(polarscale)
Dim polarGrid As PolarGrid = New PolarGrid(polarAxes, ChartObj.GRID_MAJOR)
' Create the ChartView object to place graph objects in.
Dim chartVu As ChartView = ChartView1
' Add the polar axes to the chartVu object
chartVu.AddChartObject(polarAxes)
chartVu.AddChartObject(polarGrid)
```

Antenna Grids

Class AntennaGrid

GraphObj
|
+-- Grid
|
+-- AntennaGrid

The **AntennaGrid** class defines a grid for antenna axes. The antenna grid consists of two parts: the circular grid and the radial grid. The circular grid consists of a group of concentric circles centered on the origin and aligned with the y-axis tick marks. The antenna radial grid consists of a group of radial lines, aligned with the angle tick marks and starting at the origin extending to the outer edge of the antenna coordinate system.

AntennaGrid constructors

There are two **AntennaGrid** constructors.

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal baseaxis As AntennaAxes, _
   ByVal gridtype As Integer _
)


Overloads Public Sub New( _
   ByVal baseaxis As AntennaAxes, _
   ByVal gridmagtype As Integer, _
   ByVal gridangletype As Integer _
)


[C#]
public AntennaGrid(
   AntennaAxes baseaxis,
   int gridtype
);

public AntennaGrid (
   AntennaAxes baseaxis,
   int gridmagtype,
   int gridangletype
);
```

baseaxis The antenna axes associated with the grid.

gridtype	Specifies if the radial and angular 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 antenna axis.
gridmagtype	Specifies if the radial 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 antenna axis.
gridangletype	Specifies if the angular 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 antenna axis.

The **SetLineWidth**, **SetLineStyle** and **SetColor** methods are used to customize the drawing properties of the lines used to draw the axes lines and tick marks.

Antenna grid example

[C#]

```
AntennaAxes pAntennaAxis = pAntennaTransform.GetCompatibleAxes();
pAntennaAxis.LineColor = Color.Black;
chartVu.AddChartObject(pAntennaAxis);

AntennaGrid pAntennaGrid = new AntennaGrid(pAntennaAxis, AntennaGrid.GRID_ALL);
pAntennaGrid.ChartObjAttributes = new ChartAttribute(Color.LightBlue, 1,
DashStyle.Solid);
chartVu.AddChartObject(pAntennaGrid);
```

[Visual Basic]

```
Dim pAntennaAxis As AntennaAxes = pAntennaTransform.GetCompatibleAxes()
pAntennaAxis.LineColor = Color.Black
chartVu.AddChartObject(pAntennaAxis)

Dim pAntennaGrid As New AntennaGrid(pAntennaAxis, AntennaGrid.GRID_ALL)
pAntennaGrid.ChartObjAttributes = New ChartAttribute(Color.LightBlue, 1,
DashStyle.Solid)
```

```
chartVu.AddChartObject(pAntennaGrid)
```

10. Simple Plot Objects

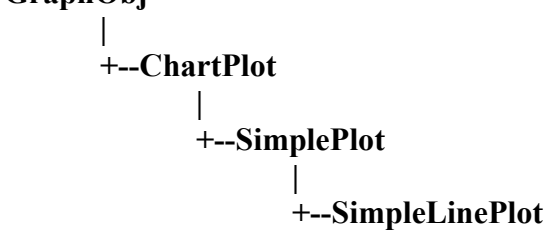
SimplePlot
- **SimpleBarPlot**
- **SimpleLineMarkerPlot**
- **SimpleLinePlot**
- **SimpleScatterPlot**
- **SimpleVersaPlot**

The **SimplePlot** class is an abstract class representing plot types that 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. 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



The **SimpleLinePlot** class is a concrete implementation of the **SimplePlot** class and and displays simple datasets in line plot format. Data points are connected using a straight line, or a step line.

SimpleLinePlot contructor

```
 [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 and fill color) for the line plot.

A **ChartAttribute** object sets the objects global line color, fill color and fill mode. Change the **ChartAttribute** object using the objects **SetChartObjAttributes** method. Use the **SetColor** method to set the color of a simple plot object. 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 line plot example (extracted from the example program BigChartDemo, class LineFill)

[C#]

```
TimeCoordinates  pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(DatasetArray,
                    ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
.
.
.

ChartAttribute attrib1 =
           new ChartAttribute (Color.Blue, 3,ChartObj.DashStyle.Solid.);
SimpleLinePlot thePlot1 =
           new SimpleLinePlot(pTransform1, Dataset1, attrib1);
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
Dim pTransform1 As TimeCoordinates = New TimeCoordinates()
pTransform1.AutoScale(DatasetArray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
.
.
.
Dim attrib1 As New ChartAttribute(Color.Blue, 3, DashStyle.Solid)
Dim thePlot1 As SimpleLinePlot = _
       New SimpleLinePlot(pTransform1, Dataset1, attrib1)
chartVu.AddChartObject(thePlot1)
```

Simple line plot example using segment colors (extracted from the example program BigChartDemo, class LineFill)

[C#]

```
TimeSimpleDataset[] DatasetArray = {Dataset1, Dataset2, Dataset3 };
pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(DatasetArray,ChartObj.AUTOAXES_FAR ,ChartObj.AUTOAXES_FAR);
pTransform1.SetGraphBorderDiagonal(0.15, .1, .92, 0.75) ;
Background background = new Background( pTransform1,
           ChartObj.GRAPH_BACKGROUND, Color.FromArgb(100,50,255),
           Color.FromArgb(40,25,120), ChartObj.Y_AXIS);
chartVu.AddChartObject(background);
Background plotbackground =
       new Background( pTransform1, ChartObj.PLOT_BACKGROUND, Color.Black);
chartVu.AddChartObject(plotbackground);
.
.  // Define and add axes, axes labels and grids to the chart
.
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 3,DashStyle.Solid);
Dataset1.SortByX(true);
thePlot1 = new SimpleLinePlot(pTransform1, Dataset1, attrib1);
chartVu.AddChartObject(thePlot1);

ChartAttribute attrib2 = new ChartAttribute (Color.Yellow, 3,DashStyle.Solid);
Dataset2.SortByX(true);
thePlot2 = new SimpleLinePlot(pTransform1, Dataset2, attrib2);
chartVu.AddChartObject(thePlot2);
```

```
ChartAttribute lossAttrib = new ChartAttribute (Color.Red, 1,DashStyle.Solid,
              Color.Red);
ChartAttribute profitAttrib =
  new ChartAttribute (Color.Green, 1,DashStyle.Solid,       Color.Green);
profitAttrib.SetFillFlag(true);
lossAttrib.SetFillFlag(true);
profitAttrib.SetLineFlag(false);
lossAttrib.SetLineFlag(false);
// Must call the linePlot (or similar function) before setting segment
// attributes so that it know size of segment buffer to allocate.
thePlot3 = new SimpleLinePlot(pTransform1, Dataset3, profitAttrib);
double []yValues = Dataset3.GetYData();
thePlot3.SetSegmentAttributesMode(true);
for (i=0; i < Dataset3.GetNumberDatapoints(); i++)
{
    if (yValues[i] > 0.0)
        thePlot3.SetSegmentAttributes(i,profitAttrib);
    else
        thePlot3.SetSegmentAttributes(i,lossAttrib);
}
chartVu.AddChartObject(thePlot3);
```

[Visual Basic]

```
Dim DatasetArray As TimeSimpleDataset() = {Dataset1, Dataset2, Dataset3}

Dim pTransform1 As TimeCoordinates = New TimeCoordinates()
pTransform1.AutoScale(DatasetArray, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
pTransform1.SetGraphBorderDiagonal(0.15, 0.1, 0.92, 0.75)
           Dim background As New Background(pTransform1,
           ChartObj.GRAPH_BACKGROUND, _ Color.FromArgb(100, 50, 255), _
           Color.FromArgb(40, 25, 120), ChartObj.Y_AXIS)
chartVu.AddChartObject(background)

Dim plotbackground As New Background(pTransform1, _
           ChartObj.PLOT_BACKGROUND, Color.Black)
chartVu.AddChartObject(plotbackground)

.
.  ' Define and add axes, axes labels and grids to the chart
.
```

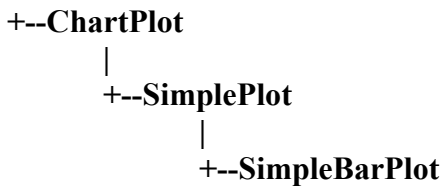
```
Dim attrib1 As New ChartAttribute(Color.Blue, 3, DashStyle.Solid)
Dataset1.SortByX(True)
Dim thePlot1 As SimpleLinePlot = _
       New SimpleLinePlot(pTransform1, Dataset1, attrib1)
chartVu.AddChartObject(thePlot1)

Dim attrib2 As New ChartAttribute(Color.Yellow, 3, DashStyle.Solid)
Dataset2.SortByX(True)
Dim thePlot2 As SimpleLinePlot = New SimpleLinePlot(pTransform1, _
                  Dataset2, attrib2)
chartVu.AddChartObject(thePlot2)


Dim lossAttrib As New ChartAttribute(Color.Red, 1, _
        DashStyle.Solid, Color.Red)
Dim profitAttrib As New ChartAttribute(Color.Green, 1, _
       DashStyle.Solid, Color.Green)
profitAttrib.SetFillFlag(True)
lossAttrib.SetFillFlag(True)
profitAttrib.SetLineFlag(False)
lossAttrib.SetLineFlag(False)
' Must call the linePlot (or similar function) before setting segment
' attributes so that it know size of segment buffer to allocate.
Dim thePlot3 As SimpleLinePlot = New SimpleLinePlot(pTransform1, _
       Dataset3, profitAttrib)
Dim yValues As Double() = Dataset3.GetYData()
thePlot3.SetSegmentAttributesMode(True)
For i = 0 To (Dataset3.GetNumberDatapoints()) - 1
  If yValues(i) > 0.0 Then
     thePlot3.SetSegmentAttributes(i, profitAttrib)
  Else
     thePlot3.SetSegmentAttributes(i, lossAttrib)
  End If
Next i
chartVu.AddChartObject(thePlot3)
```

Simple Bar Plots

Class SimpleBarPlot

GraphObj

|



The **SimpleBarPlot** class is a concrete implementation of the **SimplePlot** class and displays data in a 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.

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
);
```

transform	The coordinate system for the new **SimpleBarPlot** object.
dataset	The bar plot represents the values in this dataset.
barwidth	The width of the bars in physical coordinates.
barbase	The base value for bars in physical coordinates.
attrib	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 objects global 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 BigChartDemo, class SimpleBarChart)

[C#]

```
TimeSimpleDataset Dataset1 = new TimeSimpleDataset("Actual Sales",x1,y1);
TimeCoordinates pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR , ChartObj.AUTOAXES_FAR);
pTransform1.SetScaleStartY(0);
pTransform1.SetTimeScaleStart(new ChartCalendar(1997,ChartObj.JULY,1));
pTransform1.SetGraphBorderDiagonal(0.15, .15, .9, 0.8) ;
Background background = new Background( pTransform1, ChartObj.GRAPH_BACKGROUND,
           Color.FromArgb(30,70,70), Color.FromArgb(90,20,155), ChartObj.Y_AXIS);
chartVu.AddChartObject(background);
.
.  // Define and add axes, axes labels and grids to chart
.
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);
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
Dim Dataset1 As New TimeSimpleDataset("Actual Sales", x1, y1)
Dim pTransform1 As New TimeCoordinates()
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
pTransform1.SetScaleStartY(0)
pTransform1.SetTimeScaleStart(New ChartCalendar(1997, ChartObj.JULY, 1))

pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 0.8)
Dim background As New Background(pTransform1, _
       ChartObj.GRAPH_BACKGROUND, Color.FromArgb(30, 70, 70), _
       Color.FromArgb(90, 20, 155), ChartObj.Y_AXIS)
chartVu.AddChartObject(background)
```

```
.
.   ' Define and add axes, axes labels and grids to chart
.
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)
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 8 months.

Simple bar plot example that displays numeric data values (extracted from the example program BigChartDemo, class SimpleBarChart)

[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
|
+--SimplePlot
|
+--SimpleScatterPlot

The **SimpleScatterPlot** class is a concrete implementation of the **SimplePlot** class and displays simple datasets in scatter plot format where each data point is a symbol.

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
);
```

transform	The coordinate system for the new **SimpleScatterPlot** object.
dataset	The scatter plot represents the values in this dataset.
symtype	The symbol used in the scatter plot. Use one of the scatter plot symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE.
attrib	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.

Should you need additional symbols, create your own. Any **GraphicsPath** object can be used as a symbol. The coordinates of the symbol should assume that 1.0 is the standard symbol size with a symbol center at the relative coordinates (0.5, 0.5). The example below demonstrates how to create a diamond symbol.

```
public GraphicsPath GetDiamondShape()
{ GraphicsPath result = new GraphicsPath();
  result.AddLine(0.5f, 0.0f, 0.0f, 0.5f)  ;
  result.AddLine(0.0f, 0.5f, 0.5f, 1.0f)  ;
  result.AddLine(0.5f, 1.0f, 1.0f, 0.5f)  ;
  result.CloseFigure()  ;
  return result;
}
```

Set the custom symbol using **SimpleScatterPlot.SetCustomScatterPlotSymbol** method after the **SimpleScatterPlot** object is created.

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 BigChartDemo, class ScatterPlot)

[C#]

```
pTransform1.SetGraphBorderDiagonal(0.15, .15, .90, 0.725) ;
Background background =
    new Background( pTransform1, ChartObj.PLOT_BACKGROUND, Color.White);
```

```
chartVu.AddChartObject(background);
.
. // Define and add axes, axes labels and grids to chart
.
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid);
attrib1.SetFillColor (Color.Blue);
attrib1.SetFillFlag (true);
attrib1.SetSymbolSize(10);
SimpleScatterPlot thePlot1 =
       new SimpleScatterPlot(pTransform1, Dataset2,  ChartObj.CROSS, attrib1);
chartVu.AddChartObject(thePlot1);

ChartAttribute attrib3 = new ChartAttribute (Color.Red, 1,DashStyle.Solid);
attrib3.SetFillColor (Color.Red);
attrib3.SetFillFlag (true);
attrib3.SetSymbolSize(6);
SimpleScatterPlot thePlot3 =
     new SimpleScatterPlot(pTransform1, Dataset3,  ChartObj.CIRCLE, attrib3);
chartVu.AddChartObject(thePlot3);
```

[Visual Basic]

```
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _
           ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset3, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
pTransform1.SetGraphBorderDiagonal(0.125, 0.15, 0.95, 0.725)
Dim background As New Background(pTransform1, ChartObj.PLOT_BACKGROUND,  _
           Color.White)
chartVu.AddChartObject(background)

.
. ' Define and add axes, axes labels and grids to chart
.

Dim attrib1 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid)
attrib1.SetFillColor(Color.Blue)
attrib1.SetFillFlag(True)
attrib1.SetSymbolSize(10)
Dim thePlot1 As New SimpleScatterPlot(pTransform1, Dataset2, _
       ChartObj.CROSS, attrib1)
chartVu.AddChartObject(thePlot1)

Dim attrib2 As New ChartAttribute(Color.Green, 3, DashStyle.Solid)
```

```
Dim thePlot2 As New SimpleLinePlot(pTransform1, Dataset1, attrib2)
chartVu.AddChartObject(thePlot2)

Dim attrib3 As New ChartAttribute(Color.Red, 1, DashStyle.Solid)
attrib3.SetFillColor(Color.Red)
attrib3.SetFillFlag(True)
attrib3.SetSymbolSize(6)
Dim thePlot3 As New SimpleScatterPlot(pTransform1, Dataset3, _
       ChartObj.CIRCLE, attrib3)
chartVu.AddChartObject(thePlot3)
```

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 BigChartDemo, class ScatterPoints)

[C#]

```
.
.
.
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid);
attrib1.SetFillColor (Color.Blue);
attrib1.SetLineFlag(true);
attrib1.SetSymbolSize(10);
SimpleScatterPlot thePlot1 =
    new SimpleScatterPlot(pTransform1, Dataset1,  ChartObj.SQUARE, attrib1);
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]

```
.
.
.
Dim attrib1 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid)
```

```
attrib1.SetFillColor(Color.Blue)
attrib1.SetLineFlag(True)
attrib1.SetSymbolSize(10)
Dim thePlot1 As New SimpleScatterPlot(pTransform1, _
       Dataset1, ChartObj.SQUARE, attrib1)
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
|
+--ChartPlot
|
+--SimplePlot
|
+-- SimpleLineMarkerPlot

The **SimpleLineMarkerPlot** class is a concrete implementation of the **SimplePlot** class and displays simple datasets in a line plot format where scatter plot symbols highlight individual data points.

SimpleLineMarkerPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal transform As PhysicalCoordinates, _
   ByVal dataset As GroupDataset, _
   ByVal attribs As ChartAttribute() _
)

[C#]
public StackedLinePlot(
   PhysicalCoordinates transform,
   GroupDataset dataset,
   ChartAttribute[] attribs
);
```

transform	The coordinate system for the new **SimpleLineMarkerPlot** object.
dataset	The line marker plot represents the values in this dataset.
symtype	The symbol used in the line marker plot. Use one of the scatter plot symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE.
lineattrib	Specifies the attributes (line color) for the line part of the line marker plot.
symbolattrib	Specifies the attributes (line and fill color) for the symbol part of the line marker plot.
nsymbolstart	Specifies the starting index for symbols in the line marker plot.
nsymbolskip	Specifies the skip factor for placing symbols in the line marker plot.

A **ChartAttribute** object sets the objects global line attributes. Change the **ChartAttribute** object using the objects **SetChartObjAttributes** method. Use the objects **setSymbolAttributes** to change the attributes of the marker symbol.

Should you need additional symbols, create your own. Any **GraphicsPath** object can be used as a symbol. The coordinates of the symbol should assume that 1.0 is the standard symbol size with a symbol center at the relative coordinates (0.5, 0.5). See the example in the discussion of the **SimpleScatterPlot** class.

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 BigChartDemo, class LabeledDatapoints)

[C#]

```
SimpleDataset Dataset1 = new SimpleDataset("First",x1,y1);
CartesianCoordinates pTransform1 =
       new CartesianCoordinates( ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);

pTransform1.SetGraphBorderDiagonal(0.15, .15, .90, 0.7) ;
```

```
.
. // Define and add axes, axes labels and grids to chart
.
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid);
ChartAttribute attrib2 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid);
attrib2.SetFillColor (Color.Blue);
attrib2.SetFillFlag (true);
attrib2.SetSymbolSize(10);
SimpleLineMarkerPlot thePlot1 =
               new SimpleLineMarkerPlot(pTransform1, Dataset1,
               ChartObj.SQUARE, attrib1,attrib2, 0, 1);
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
Dim Dataset1 As New SimpleDataset("First", x1, y1)
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _
       ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)

pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 0.7)
.
. ' Define and add axes, axes labels and grids to chart
.

Dim attrib1 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid)
Dim attrib2 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid)
attrib2.SetFillColor(Color.Blue)
attrib2.SetFillFlag(True)
attrib2.SetSymbolSize(10)
Dim thePlot1 As New SimpleLineMarkerPlot(pTransform1, Dataset1, _
       ChartObj.SQUARE, 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#]

```
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]

```
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)
chartVu.AddChartObject(thePlot1)
```

Simple Versa Plots

Class SimpleVersaPlot

GraphObj
|
+--ChartPlot
|
+--SimplePlot
|
+-- SimpleLineMarkerPlot
|
+-- SimpleVersaPlot

The **SimpleVersaPlot** is a plottype that can be any of the four simple plot types: LINE_MARKER_PLOT, LINE_PLOT, BAR_PLOT, SCATTER_PLOT. It is used when you want to be able to change from one plot type to another, without deleting the instance of the old plot object and creating an instance of the new.

SimpleLineMarkerPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal PhysicalCoordinates As PhysicalCoordinates, _
   ByVal SimpleDataset As SimpleDataset, _
   ByVal ChartAttribute As ChartAttribute _
)

Overloads Public Sub New( _
   ByVal PhysicalCoordinates As PhysicalCoordinates, _
   ByVal SimpleDataset As SimpleDataset, _
   ByVal Double As Double, _
   ByVal Double As Double, _
   ByVal ChartAttribute As ChartAttribute, _
   ByVal Int32 As Integer _
)

Overloads Public Sub New( _
   ByVal PhysicalCoordinates As PhysicalCoordinates, _
   ByVal SimpleDataset As SimpleDataset, _
   ByVal Int32 As Integer, _
   ByVal ChartAttribute As ChartAttribute _
)


Overloads Public Sub New( _
   ByVal PhysicalCoordinates As PhysicalCoordinates, _
   ByVal SimpleDataset As SimpleDataset, _
   ByVal Int32 As Integer, _
   ByVal ChartAttribute As ChartAttribute, _
   ByVal ChartAttribute As ChartAttribute, _
   ByVal Double As Double _
)

[C#]


public SimpleVersaPlot(
   PhysicalCoordinates transform,
   SimpleDataset dataset,
   ChartAttribute lineattrib
);


public SimpleVersaPlot(
   PhysicalCoordinates transform,
   SimpleDataset dataset,
   Double barwidth,
   Double barbase,
   ChartAttribute attrib,
   Int32 barjust
);


public SimpleVersaPlot(
   PhysicalCoordinates transform,
   SimpleDataset dataset,
   Int32 symtype,
   ChartAttribute symbolattrib
);

public SimpleVersaPlot(
   PhysicalCoordinates transform,
   SimpleDataset dataset,
   Int32 symtype,
   ChartAttribute lineattrib,
   ChartAttribute symbolattrib,
   Double barwidth
);
```

transform The coordinate system for the new **SimpleVersaPlot** object.

dataset	The versa plot represents the values in this dataset.
symtype	The symbol used in the line marker plot. Use one of the scatter plot symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE.
lineattrib	Specifies the attributes (line color and line style) for the line part of the line marker plot.
symbolattrib	Specifies the attributes (line and fill color) for the symbol part of the line marker plot.
nsymbolstart	Specifies the starting index for symbols in the line marker plot.
nsymbolskip	Specifies the skip factor for placing symbols in the line marker plot.
barwidth	Specifies the width of the bar.

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.

Change the plot type using the PlotType property. Use one of the plot type constants: LINE_MARKER_PLOT, LINE_PLOT, BAR_PLOT, SCATTER_PLOT.

Simple versa-plot example (extracted from the example program NewDemosRev2.SimpleVersaChart)

[C#]

```
ChartAttribute attrib1 = new ChartAttribute(Color.Green, 0, DashStyle.Solid,
Color.Green);


attrib1.SetFillFlag(true);
thePlot1 = new SimpleVersaPlot(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)
thePlot1 = New SimpleVersaPlot(pTransform1, Dataset1, _
    ChartCalendar.GetCalendarWidthValue(ChartObj.MONTH, 8), 0.0R, 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)
```

11. Group Plot Objects

GroupPlot
- **ArrowPlot**
- **BubblePlot**
- **CandlestickPlot**
- **CellPlot**
- **ErrorBarPlot**
- **FloatingBarPlot**
- **GroupBarPlotChartPlot**
- **HistogramPlot**
- **LineGapPlot**
- **MultiLinePlot**
- **OHLCPlot**
- **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
|
+--ChartPlot
|
+--GroupPlot
|
+--ArrowPlot

The **ArrowPlot** class is a concrete implementation 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 numer of groups of the group dataset must be three.

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 numer of groups must be three. Orgainize 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.

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 BigChartDemo, class ArrowPlotChart)

[C#]

```
GroupDataset Dataset1 = new GroupDataset("First",x1,y1);
Dataset1.SetAutoScaleNumberGroups(1);
CartesianCoordinates pTransform1 =
new CartesianCoordinates( ChartObj.LINEAR_SCALE, ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
pTransform1.SetScaleX(0,10);
pTransform1.SetScaleY(0,10);
pTransform1.SetGraphBorderDiagonal(0.15, .15, .90, 0.75) ;
.
.       // Define axes, axes labels and grids
.
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.SetAutoScaleNumberGroups(1)
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _
       ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
pTransform1.SetScaleX(0, 10)
pTransform1.SetScaleY(0, 10)
pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 0.75)
Dim background As New Background(pTransform1, _
       ChartObj.PLOT_BACKGROUND, Color.White)
chartVu.AddChartObject(background)

.
.       ' Define axes, axes labels and grids
.

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.

Box and Whisker Plots

Class BoxWhiskerPlot

GraphObj
|
+--ChartPlot
|
+--GroupPlot
|
+-- BoxWhiskerPlot

The **BoxWhiskerPlot** class extends the **GroupPlot** class and displays statistical data in a box and whisker format. The **BoxWhiskerPlot** class graphically represents groups of numerical data through their five-number summaries (the smallest observation, lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation). A **BoxWhiskerPlot** is unique among our chart types because the data is not represented by a 1D or 2D matrix. Instead, it consists of multiple populations, where each population can have a different number of data points. Each population is summarized by 5 statistics (the smallest observation, lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation), display graphically in a chart. Read the Wikipedia entry for more information concerning box and whisker plots: http://en.wikipedia.org/wiki/Box_plot

BoxWhiskerPlot constructor

```
Visual Basic (Declaration)
Public Sub New ( _
    transform As PhysicalCoordinates, _
    rwidth As Double, _
    attrib As ChartAttribute _
)

C#
public BoxWhiskerPlot(
    PhysicalCoordinates transform,
    double rwidth,
    ChartAttribute attrib
)
```

transform	The coordinate system for the new **BoxWhiskerPlot** object.
rwidth	The width of the candlestick box in physical coordinates.
attrib	Specifies the attributes (line color and fill color) of the candlestick lines when the close value is greater than the open value.

Once the initial **BoxWhiskerPlot** object is created, populations are added to it, one population at a time, using the **AddPopulation** method. Each population can have a different number of data points. Once all of the population groups are added, the software will calculate the quartile data for each population in response to the **AutoBWChart** method call. Each population is summarized by a single box in the box and whisker plot.

Visual Basic (Declaration)

```
Public Sub AddPopulation ( _
    pop As Double(), _
    xvalue As Double _
)
```

```
C#
public void AddPopulation(
    double[] pop,
    double xvalue
)
```

Parameters

pop	The source population of y-values to add.
xvalue	The x-value of the of y-values population.

There are several variants of box and whisker plots. Select which one you want using the **BWFormat** property. Use one of the BW format constants: BW_MINMAX_WHISKER, BW_IRQ15_WHISKER_OUTLIERS, BW_IQR15_WHISKER_ALLPOINTS.

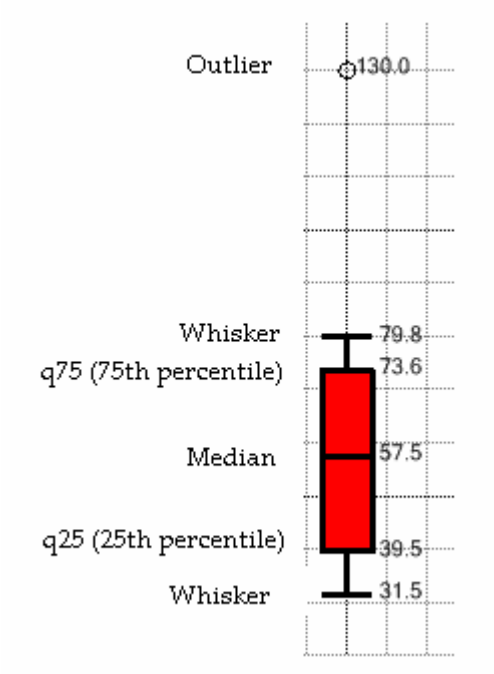
BW_MINMAX_WHISKER	Plot minimum, maximum values as whiskers, and the median, q25 and q75 values as the box.
BW_IQR15_WHISKER_OUTLIERS	Plot the minimum value within q25 - 1.5*IQR and maximum value within q75 + 1.5*IQR as whiskers, the median, q25 and q75 values as the box, and outliers as scatter plot symbols.
BW_IQR15_WHISKER_ALLPOINTS	Plot the minimum value within q25 - 1.5*IQR and maximum value within q75 + 1.5*IQR as the whiskers, the median, q25 and q75 values as the box, and plot all points as scatter plot symbols.

Where

q25 for a given population, q25 is the 25th percentile point in the population

q75 for a given population, q75 is the 75th percentile point in the population

IQR for a given population, IQR is the value q75 - q25.



Turn on the numeric labeling of the Box and Whisker summary values (high whisker, q75, median, q25, low whisker) by setting the **BoxWhiskerPlot**. **ShowDatapointValue** property true. Turn on the numeric labeling of the scatter plot symbols used for outliers by setting the the **BoxWhiskerPlot**. **ScatterPlot.ShowDatapointValue** property true. You should not combine numeric labeling with the W_IQR15_WHISKER_ALLPOINTS option, unless you are working with a very small number of data points; otherwise the labels will all overlap one another.

Box and whisker plot example (extracted from the example program NewDemosRev2.BoxAndWhiskerChart)

[C#]

```
            //New York City

double[] NYCity = { 31.5, 33.6, 42.4, 52.5, 62.7, 71.6, 130, 76.8, 75.5, 68.2,
57.5, 47.6, 36.6 };

            //Houston

double[] Houston = { 50.4, 53.9, 60.6, 68.3, 74.5, 80.4, 5, 100, 82.6, 82.3, 78.2,
69.6, 61, 53.5 };

            //San Francisco

double[] SanFrancisco = { 48.7, 52.2, 53.3, 55.6, 58.1, 76, 61.5, 62.7, 63.7,
64.5, 61, 54.8, 49.4 };

            //Bouston
```

```
double[] Boston = { 32.4, 53.9, 44.6, 58.3, 64.5, 70.4, 73, 90, 72.6, 72.3, 68.2,
49.6, 41, 33.5 };

            //Pittsburgh

double[] Pittsburgh = { 41.4, 54, 24.6, 38.3, 44.5, 61.4, 63, 105, 72.6, 72.3,
68.2, 49.6, 41, 33.5 };

double minval = 0.0, maxval = 0.0;

int i;

int numpnts = NYCity.Length + Houston.Length + SanFrancisco.Length + Boston.Length
+ Pittsburgh.Length;

.

.

.

ChartAttribute defaultattrib = new ChartAttribute(Color.Black, 1, DashStyle.Solid,
Color.Red);

defaultattrib.SetFillFlag(true);

ChartAttribute fillattrib = new ChartAttribute(Color.Black, 3, DashStyle.Solid,
Color.Red);

fillattrib.SetFillFlag(true);

BoxWhiskerPlot thePlot1 = new BoxWhiskerPlot(pTransform1, 0.25, fillattrib);

thePlot1.AddPopulation(NYCity, 1);

thePlot1.AddPopulation(Houston, 2);

thePlot1.AddPopulation(SanFrancisco, 3);

thePlot1.AddPopulation(Boston, 4);

thePlot1.AddPopulation(Pittsburgh, 5);

thePlot1.BWFormat = 1;

thePlot1.BarDatapointLabelPosition = ChartObj.CENTERED_BAR;

thePlot1.PlotLabelTemplate.TextFont = new Font("Microsoft Sans Serif", 8,
FontStyle.Regular);

thePlot1.PlotLabelTemplate.DecimalPos = 1;

thePlot1.ShowDatapointValue = true;

// label outliers

thePlot1.ScatterPlot.ShowDatapointValue = true;

thePlot1.AutoBWChart();


chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
'New York City

Dim NYCity As Double() = {31.5, 33.6, 42.4, 52.5, 62.7, 71.6, _

        130, 79.8, 75.5, 68.2, 57.5, 47.6, _

        36.6}
```

```
'Houston
Dim Houston As Double() = {50.4, 53.9, 60.6, 68.3, 74.5, 80.4, _
        5, 100, 82.6, 82.3, 78.2, 69.6, _
        61, 53.5}

'San Francisco
Dim SanFrancisco As Double() = {48.7, 52.2, 53.3, 55.6, 58.1, 76, _
        61.5, 62.7, 63.7, 64.5, 61, 54.8, _
        49.4}



'Boston

Dim Boston As Double() = {32.4, 53.9, 44.6, 58.3, 64.5, 70.4, _
        73, 90, 72.6, 72.3, 68.2, 49.6, _
        41, 33.5}

'Pittsburgh

Dim Pittsburgh As Double() = {41.4, 54, 24.6, 38.3, 44.5, 61.4, _
        63, 105, 72.6, 72.3, 68.2, 49.6, _
        41, 33.5}
.
.
.
Dim defaultattrib As New ChartAttribute(Color.Black, 1, DashStyle.Solid,
Color.Red)
defaultattrib.SetFillFlag(True)
Dim fillattrib As New ChartAttribute(Color.Black, 3, DashStyle.Solid, Color.Red)
fillattrib.SetFillFlag(True)
thePlot1 = New BoxWhiskerPlot(pTransform1, 0.25, fillattrib)
thePlot1.AddPopulation(NYCity, 1)
thePlot1.AddPopulation(Houston, 2)
thePlot1.AddPopulation(SanFrancisco, 3)
thePlot1.AddPopulation(Boston, 4)
thePlot1.AddPopulation(Pittsburgh, 5)

thePlot1.BWFormat = ChartObj.BW_IQR15_WHISKER_OUTLIERS
thePlot1.BarDatapointLabelPosition = ChartObj.CENTERED_BAR
thePlot1.PlotLabelTemplate.TextFont = _
    New Font("Microsoft Sans Serif", 8, FontStyle.Regular)

thePlot1.PlotLabelTemplate.DecimalPos = 1
```

```
thePlot1.ShowDatapointValue = True
' label outliers
thePlot1.ScatterPlot.ShowDatapointValue = True
thePlot1.AutoBWChart()

 chartVu.AddChartObject(thePlot1)
```

Bubble Plots

Class BubblePlot

GraphObj
|
+--ChartPlot
|
+--GroupPlot
|
+--BubblePlot

The **BubblePlot** class is a concrete implementation of the **GroupPlot** class. It displays bubble plots. A group dataset specifies the position and size of each bubble in a bubble plot. The numer of groups must be two.

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
);
```

transform	The coordinate system for the new bubble plot object.
dataset	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).

bubblesizetype	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.
attrib	Specifies the attributes (line color and fill color) of the bubble plot circles.

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 BigChartDemo, class BubblePlotChart)

[C#]

```
TimeGroupDataset Dataset1 = new TimeGroupDataset("First",x1,y1);
Dataset1.SetStackMode(ChartObj.AUTOAXES_STACKED);
TimeCoordinates pTransform1 =
new TimeCoordinates( ChartObj.TIME_SCALE, ChartObj.LINEAR_SCALE);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
pTransform1.SetGraphBorderDiagonal(0.15, .15, .80, 0.75) ;
.
.  // Define axes, axes labels and grids
.
ChartAttribute attrib1 = new ChartAttribute (Color.Black, 0,DashStyle.Solid);
attrib1.SetFillColor (Color.FromArgb(177, 33, 33));
attrib1.SetFillFlag (true);
BubblePlot thePlot1 = new BubblePlot(pTransform1, Dataset1,
                          ChartObj.SIZE_BUBBLE_RADIUS, attrib1);
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
Dim Dataset1 As New TimeGroupDataset("First", x1, y1)
Dataset1.SetStackMode(ChartObj.AUTOAXES_STACKED)
Dim pTransform1 As New TimeCoordinates(ChartObj.TIME_SCALE, ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.8, 0.75)
.
```

```
.   ' Define axes, axes labels and grids
.

Dim attrib1 As New ChartAttribute(Color.Black, 0, DashStyle.Solid)
attrib1.SetFillColor(Color.FromArgb(177, 33, 33))
attrib1.SetFillFlag(True)
Dim thePlot1 As New BubblePlot(pTransform1, Dataset1, _
       ChartObj.SIZE_BUBBLE_RADIUS, attrib1)
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
|
+--ChartPlot
|
+--GroupPlot
|
+-- CandlestickPlot

The **CandlestickPlot** class is a concrete implementation 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.

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. Orgainize 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.
rwidth	The width of the candlestick box in physical coordinates.
defaultattrib	Specifies the default attributes (line color and fill color) of the candlestick lines and box.
fillattrib	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 BigChartDemo, class CandlestickFinPlot)

[C#]

```
TimeGroupDataset Dataset1 =
        new TimeGroupDataset("Stock Data",xValues,stockPriceData);
```

```
.
.  // Define axes, axes labels and grids
.
ChartAttribute defaultattrib =
        new ChartAttribute(Color.Black, 1,DashStyle.Solid, Color.White);
defaultattrib.SetFillFlag(true);
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);
```

[Visual Basic]

```
Dim Dataset1 As New TimeGroupDataset("Stock Data", xValues, stockPriceData)
.
.  ' Define axes, axes labels and grids
.
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
|
+--ChartPlot
|
+--GroupPlot

```
|
+--CellPlot
```

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.

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
);
```

transform	The coordinate system for the new **CellPlot** object.
dataset	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.
attrib	Specifies the attributes (line and fill color) 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 BigChartDemo, class CellPlotChart)

[C#]

```
GroupDataset Dataset1 = new GroupDataset("First",x1,y1);
Dataset1.SetAutoScaleNumberGroups(1); // picks up on width, but because data is
                                      // 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);
pTransform1.SetGraphBorderDiagonal(0.15, .15, .90, 0.75) ;
.
.  // Define axes, axes labels and grids
.
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 1,DashStyle.Solid);
attrib1.SetFillColor (Color.Blue);
attrib1.SetFillFlag (true);
CellPlot thePlot1 = new CellPlot(pTransform1, Dataset1,  attrib1);
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]

```
Dim Dataset1 As New GroupDataset("First", x1, y1)

Dataset1.SetAutoScaleNumberGroups(1) ' picks up on width, but because data is
                                     ‘ random, should still work
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _
       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(maxx)
pTransform1.SetScaleStopY(maxy)
' Re-auto-scale to produce rounded axis values.
pTransform1.AutoScale(ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR)
.
.  ' Define axes, axes labels and grids
.

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)
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
|
+--ChartPlot
|
+--GroupPlot
|
+-- ErrorBarPlot

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.

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
);
```

transform	The coordinate system for the new **ErrorBarPlot** object. The number of groups must be two.
dataset	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.
rbarwidth	The width of the error bars.
attrib	Specifies the attributes (line color) 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

|

```
+--ChartPlot
       |
       +--GroupPlot
              |
              +-- FloatingBarPlot
```

The **FloatingBarPlot** class extends the **FloatingBarPlot** 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.

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.
attrib	Specifies the attributes (line and fill color) for the floating bars.
nbarjust	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 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 BigChartDemo, class FloatingBarChart)

[C#]

```
GroupDataset Dataset1 =
```

```
   new GroupDataset("Actual Sales",x1,y1);
CartesianCoordinates pTransform1 = new CartesianCoordinates();
pTransform1.SetScaleStartX(0);
pTransform1.SetScaleStartY(0);
pTransform1.SetScaleStopX(12);
pTransform1.SetScaleStopY(7);
.
.  // Define axes, axes labels and grids
.
FloatingBarPlot thePlot1 = new FloatingBarPlot(pTransform1);
ChartAttribute attrib1 =
   new ChartAttribute (Color.Black, 1,ChartObj.DashStyle.Solid., Color.Green);
attrib1.SetFillFlag(true);
thePlot1.floatingBarPlot(Dataset1, 0.75, attrib1, ChartObj.JUSTIFY_CENTER);
thePlot1.SetBarOrient(ChartObj.HORIZ_DIR);
```

[Visual Basic]

```
Dim Dataset1 As New GroupDataset("Actual Sales", x1, y1)
Dim pTransform1 As New CartesianCoordinates()

pTransform1.SetScaleStartX(0)
pTransform1.SetScaleStartY(0)
pTransform1.SetScaleStopX(12)
pTransform1.SetScaleStopY(7)
.
.  ' Define axes, axes labels and grids
.
Dim attrib1 As New ChartAttribute(Color.Black, 1, DashStyle.Solid, Color.Green)
attrib1.SetFillFlag(True)
Dim thePlot1 As New FloatingBarPlot(pTransform1, Dataset1, 0.75, attrib1, _
       ChartObj.JUSTIFY_CENTER)
thePlot1.SetBarOrient(ChartObj.HORIZ_DIR)
```

Floating bar plot example that uses date/time values as the floating bar endpoints (extracted from the example program BigChartDemo, class FloatingBarChart, method InitializeChart2)

[C#]

```
int nnumpnts = 18;
int numgroups = 2;
```

```
double []x1= new double[nnumpnts];
ChartCalendar [,]y1 = new ChartCalendar[numgroups,nnumpnts];

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[17] = 1;
y1[0,17] = new ChartCalendar(2002, ChartObj.JULY,1);
y1[1,17] = new ChartCalendar(2002, ChartObj.OCTOBER,1);

theFont = new Font("Microsoft Sans Serif", 10,  FontStyle.Bold);
TimeGroupDataset Dataset1 = new TimeGroupDataset("Actual Sales",x1,y1);
TimeCoordinates pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(Dataset1,ChartObj.AUTOAXES_FAR,  ChartObj.AUTOAXES_NEAR);

pTransform1.SwapScaleOrientation();
pTransform1.SetScaleStartY(0);
pTransform1.SetGraphBorderDiagonal(0.22, .15, .95, 0.8) ;
.
.  // Define axes, axes labels and grids
.
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 nnumpnts As Integer = 18
Dim numgroups As Integer = 2
Dim x1(nnumpnts - 1) As Double
Dim y1(numgroups - 1, nnumpnts - 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(17) = 1
y1(0, 17) = New ChartCalendar(2002, ChartObj.JULY, 1)
y1(1, 17) = New ChartCalendar(2002, ChartObj.OCTOBER, 1)


theFont = New Font("Microsoft Sans Serif", 10, FontStyle.Bold)

Dim Dataset1 As TimeGroupDataset = New TimeGroupDataset("Actual Sales", x1, y1)
Dim pTransform1 As TimeCoordinates = New TimeCoordinates()
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_NEAR)

pTransform1.SwapScaleOrientation()
pTransform1.SetScaleStartY(0)
pTransform1.SetGraphBorderDiagonal(0.22, 0.15, 0.95, 0.8)
.
.  ' Define axes, axes labels and grids
.

Dim thePlot1 As FloatingBarPlot = New FloatingBarPlot(pTransform1)
Dim attrib1 As ChartAttribute = 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)
```

Floating Stacked Bar Plots

Class FloatingStackedBarPlot

GraphObj

|

+--ChartPlot

|

+--GroupPlot

```
|
+-- FloatingStackedBarPlot
```

The **FloatingStackedBarPlot** class extends the **GroupPlot** class and displays floating stacked 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 starting value for each stacked bar is Y[0]. Each bar after that is defined by the succeeding value in the group value array (Y[1], Y[2]..). Unlike the **StackedBarPlot** plot, the displayed values are not a cumulative sum of the group values. All bars in a given **FloatingStackedBarPlot** object have the same width.

FloatingStackedBarPlot 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 FloatingStackedBarPlot(
   PhysicalCoordinates transform,
   GroupDataset dataset,
   double rbarwidth,
   ChartAttribute[] attrib,
   int nbarjust
);
```

transform	The coordinate system for the new **FloatingStackedBarPlot** object.
dataset	The starting value for each stacked bar is Y[0]. Each bar after that is defined by the succeeding value in the group value array (Y[1], Y[2]..).
rbarwidth	The width of the floating bars in units of the independent axis.
attrib	An array of **ChartAttribute** specifying the color for each bar. The number of colors, and the length of the array should be one less than the number of groups in the source dataset..
nbarjust	Specifies the justification with respect to the independent data value. Use one of the justification constants: JUSTIFY_MIN, JUSTIFY_CENTER, JUSTIFY_MAX.

FloatingStackedBarPlot plots can be used in place of **OHLCPlots**, or **CandlestickPlots** for the display of financial information. See the example below.

Floating stacked bar plot example for displaying stock data. Extracted from the NewDemosRev2. FloatingStackedBars example.

[C#]

```
TimeGroupDataset Dataset1 = new TimeGroupDataset("Stock
Data",xValues,stockPriceData);
pTransform1 = new TimeCoordinates();
pTransform1.SetWeekType(weekmode);
pTransform1.AutoScale(Dataset1,ChartObj.AUTOAXES_NEAR,  ChartObj.AUTOAXES_NEAR);
pTransform1.SetGraphBorderDiagonal(0.15, .15, .82, 0.75) ;

SetInitialDates(pTransform1);

Background graphbackground1 =
    new Background( pTransform1, ChartObj.GRAPH_BACKGROUND,
    Color.White, Color.LightGray, ChartObj.Y_AXIS);
chartVu.AddChartObject(graphbackground1);
Background plotbackground1 =
    new Background( pTransform1, ChartObj.PLOT_BACKGROUND,Color.White);
chartVu.AddChartObject(plotbackground1);
.
.
.
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 };
attrib1.SetFillFlag(true);
thePlot1 = new FloatingStackedBarPlot(pTransform1,
    Dataset1, ChartCalendar.GetCalendarWidthValue(ChartObj.DAY_OF_YEAR, 0.75),
    attribArray, ChartObj.JUSTIFY_CENTER);
thePlot1.SetFastClipMode(ChartObj.FASTCLIP_X);
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
Dim Dataset1 As New TimeGroupDataset("Stock Data", xValues, stockPriceData)
pTransform1 = New TimeCoordinates()
pTransform1.SetWeekType(weekmode)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR)
pTransform1.SetGraphBorderDiagonal(0.15, .15, .82, 0.75)

SetInitialDates(pTransform1)

Dim graphbackground1 As New Background(pTransform1, ChartObj.GRAPH_BACKGROUND, _
Color.White, Color.LightGray, ChartObj.Y_AXIS)
chartVu.AddChartObject(graphbackground1)
Dim plotbackground1 As New Background(pTransform1, ChartObj.PLOT_BACKGROUND, _
Color.White)
.
.
.
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}
attrib1.SetFillFlag(True)
thePlot1 = New FloatingStackedBarPlot(pTransform1, Dataset1, _
  ChartCalendar.GetCalendarWidthValue(ChartObj.DAY_OF_YEAR, 0.75), _
  attribArray, ChartObj.JUSTIFY_CENTER)
thePlot1.SetFastClipMode(ChartObj.FASTCLIP_X)
chartVu.AddChartObject(thePlot1)
```

Group Bar Plots

Class GroupBarPlot

GraphObj
|
+--ChartPlot
|
+--GroupPlot
|
+--GroupBarPlot

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.

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 attribs As ChartAttribute(), _
   ByVal nbarjust As Integer _
)

[C#]
public GroupBarPlot(
   PhysicalCoordinates transform,
   GroupDataset dataset,
   double rbarwidth,
   double rbarbase,
   ChartAttribute[] attribs,
   int nbarjust
);
```

transform	The coordinate system for the new **GroupPlot** object.
dataset	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.
rbarwidth	The width of the group bars in units of the independent axis. All bars within a group are squeezed into the width defined by *rbarwidth*. Each individual bar within the group has a width of *rbarwidth*/*dataset*.GetNumberGroups().
rbarbase	The group bars start at the value *rbarbase*, and extend to the group bar values represented by the dataset.
attribs	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.
nbarjust	The group bars are justified with respect to the x-values in the dataset using the *rbarjust* 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 BigChartDemo, class GroupBarPlotChart)

[C#]

```
TimeGroupDataset Dataset1 =
       new TimeGroupDataset("GroupTimeData",xValues,groupBarData);
TimeCoordinates pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(Dataset1,ChartObj.AUTOAXES_NEAR,  ChartObj.AUTOAXES_NEAR);
pTransform1.SetTimeScaleStart(new ChartCalendar(1997,ChartObj.JANUARY,1));
pTransform1.SetTimeScaleStop(new ChartCalendar(2003,ChartObj.JANUARY,1));
pTransform1.SetGraphBorderDiagonal(0.1, .1, .45, 0.75) ;
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 axes, axes labels and grids
.
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};
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]

```
Dim Dataset1 As New TimeGroupDataset("GroupTimeData", xValues, groupBarData)
 ' Group Bargraph
Dim pTransform1 As New TimeCoordinates()
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR)

pTransform1.SetTimeScaleStart(New ChartCalendar(1997, ChartObj.JANUARY, 1))
```

```
pTransform1.SetTimeScaleStop(New ChartCalendar(2003, ChartObj.JANUARY, 1))

pTransform1.SetGraphBorderDiagonal(0.1, 0.1, 0.45, 0.75)

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)

pTransform1.SetScaleStartY(0)
Dim xAxis1 As New TimeAxis(pTransform1)
xAxis1.SetColor(Color.White)
chartVu.AddChartObject(xAxis1)

Dim yAxis1 As New LinearAxis(pTransform1, ChartObj.Y_AXIS)
yAxis1.SetColor(Color.White)
chartVu.AddChartObject(yAxis1)

Dim xAxisLab1 As New TimeAxisLabels(xAxis1)
xAxisLab1.SetAxisLabelsFormat(ChartObj.TIMEDATEFORMAT_Y2000)
xAxisLab1.SetColor(Color.White)
chartVu.AddChartObject(xAxisLab1)

Dim yAxisLab1 As New NumericAxisLabels(yAxis1)
yAxisLab1.SetAxisLabelsFormat(ChartObj.CURRENCYFORMAT)
yAxisLab1.SetColor(Color.White)
chartVu.AddChartObject(yAxisLab1)

Dim xgrid1 As New Grid(xAxis1, yAxis1, ChartObj.X_AXIS, ChartObj.GRID_MAJOR)
xgrid1.SetColor(Color.White)
chartVu.AddChartObject(xgrid1)

Dim ygrid1 As New Grid(xAxis1, yAxis1, ChartObj.Y_AXIS, ChartObj.GRID_MAJOR)
ygrid1.SetColor(Color.White)
chartVu.AddChartObject(ygrid1)



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}
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
|
+--ChartPlot
|
+--GroupPlot
|
+--HistogramPlot

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.

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.

rbarbase	The histogram bars start at the value *rbarbase*, and extend to the histogram bar values represented by the dataset.
attrib	Specifies the attributes (line and fill color) 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 BigChartDemo, class HistogramChart)

[C#]

```
int nnumpnts = 6;
int numgroups = 2;
double []x1= new double[nnumpnts];
double [,]y1 = new double[numgroups,nnumpnts];

//            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;

theFont = new Font("Microsoft Sans Serif", 10,  FontStyle.Bold);

GroupDataset Dataset1 = new GroupDataset("Actual Sales",x1,y1);
CartesianCoordinates pTransform1 = new CartesianCoordinates();

pTransform1.SetScaleStartY(0);
pTransform1.SetScaleStartX(0);
pTransform1.SetScaleStopX(80);
pTransform1.SetScaleStopY(1.00);
pTransform1.SetGraphBorderDiagonal(0.15, .15, .9, 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);

Background plotbackground = new Background( pTransform1,
            ChartObj.PLOT_BACKGROUND, Color.Black);
chartVu.AddChartObject(plotbackground);
.
.  // Define axes, axes labels and grids
.
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();
bardatavalue.SetTextFont(theFont);
bardatavalue.SetNumericFormat(ChartObj.PERCENTFORMAT);
bardatavalue.SetColor(Color.Black);
thePlot1.SetBarDatapointLabelPosition(ChartObj.INSIDE_BAR);
thePlot1.SetPlotLabelTemplate(bardatavalue);
thePlot1.SetShowDatapointValue(true);

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]

```
Dim Dataset1 As New GroupDataset("Actual Sales", x1, y1)
Dim pTransform1 As New CartesianCoordinates()
pTransform1.SetScaleStartY(0)
pTransform1.SetScaleStartX(0)
pTransform1.SetScaleStopX(80)
pTransform1.SetScaleStopY(1.0)
```

```
pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 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)
.
.  ' Define axes, axes labels and grids
.
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()
bardatavalue.SetTextFont(theFont)
bardatavalue.SetNumericFormat(ChartObj.PERCENTFORMAT)
bardatavalue.SetColor(Color.Black)
thePlot1.SetBarDatapointLabelPosition(ChartObj.INSIDE_BAR)
thePlot1.SetPlotLabelTemplate(bardatavalue)
thePlot1.SetShowDatapointValue(True)

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
|
+--ChartPlot
|
+--GroupPlot
|
+-- LineGapPlot

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.

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
);
```

transform	The coordinate system for the new **LineGapPlot** object.
dataset	The line gap plot represents the values in this group dataset. The number of groups in this group dataset must be two.
attrib	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 BigChartDemo, class LineGapChart)

[C#]

```
int nNumPnts = 5, nNumGroups = 2;
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] = 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;

TimeGroupDataset Dataset1 =
       new TimeGroupDataset("GroupTimeData",xValues,groupBarData);

TimeCoordinates pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(Dataset1,ChartObj.AUTOAXES_NEAR,  ChartObj.AUTOAXES_FAR);

pTransform1.SetGraphBorderDiagonal(0.15, .1, .95, 0.8) ;
Background background = new Background( pTransform1, ChartObj.GRAPH_BACKGROUND,
Color.FromArgb(0,120,70), Color.FromArgb(0,40,30), ChartObj.Y_AXIS);
chartVu.AddChartObject(background);
.
.  // Define axes, axes labels and grids
.
ChartAttribute attrib1 =
       new ChartAttribute(Color.Black, 1,DashStyle.Solid, Color.Red);
attrib1.SetFillFlag(true);
attrib1.SetLineFlag(false);
LineGapPlot thePlot1 = new LineGapPlot(pTransform1, Dataset1, attrib1);
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)
pTransform1.SetGraphBorderDiagonal(0.15, 0.1, 0.95, 0.8)
Dim background As New Background(pTransform1, ChartObj.GRAPH_BACKGROUND, _
       Color.FromArgb(0, 120, 70), Color.FromArgb(0, 40, 30), ChartObj.Y_AXIS)
chartVu.AddChartObject(background)
.
.  ' Define axes, axes labels and grids
.
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
|
+--ChartPlot
|
+--GroupPlot
|

+-- MultiLinePlot

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.

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
);
```

transform	The coordinate system for the new **MultiLinePlot** object.
dataset	The multi-line plot represents the values in this group dataset.
attribs	An array of **ChartAttribute** objects, sized the same as the number of groups in the dataset,t specify the attributes (line color) for each group of the multi-line plot.

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.

Multi-line plot example (extracted from the example program BigChartDemo, class MultiLine)

[C#]

```
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));
}
GroupDataset Dataset1 = new GroupDataset("First",x1,y1);

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, 0.70) ;
.
.       // Define axes, axes labels and grids
.
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 3,DashStyle.Solid);
ChartAttribute []attribArray = new ChartAttribute[numGroups];
for (i=0; i < numGroups; i++)
    attribArray[i] = (ChartAttribute) attrib1.Clone();
MultiLinePlot thePlot1 = new MultiLinePlot(pTransform1, Dataset1,  attribArray);
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
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

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
y1(0, 5) = ChartObj.rBadDataValue
y1(3, 15) = ChartObj.rBadDataValue
theFont = New Font("Microsoft Sans Serif", 10, FontStyle.Bold)
Dim Dataset1 As New GroupDataset("First", x1, y1)

Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _
       ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
pTransform1.SetScaleStartX(0)
pTransform1.SetScaleStartY(0)

Dim background As New Background(pTransform1, ChartObj.PLOT_BACKGROUND, _
       Color.FromArgb(255, 255, 255))
chartVu.AddChartObject(background)
.
.       ' Define axes, axes labels and grids
.

Dim attrib1 As New ChartAttribute(Color.Blue, 3, DashStyle.Solid)
Dim attribArray(numGroups) As ChartAttribute
For i = 0 To numGroups - 1
   attribArray(i) = attrib1.Clone()
Next i
Dim thePlot1 As New MultiLinePlot(pTransform1, Dataset1, attribArray)
chartVu.AddChartObject(thePlot1)
```

Open-High-Low-Close Plots

Class OHLCPlot

GraphObj
|
+--ChartPlot
|
+--GroupPlot
|
+-- OHLCPlot

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.

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
);
```

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.
rflagwidth	The width of the open and close markers in units of the independent axis.
attrib	Specifies the attributes (line color) 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 BigChartDemo, class OHLCFinPlot)

[C#]

```
TimeGroupDataset Dataset1 =
         new TimeGroupDataset("Stock Data",xValues,
                              stockPriceData);
TimeCoordinates pTransform1 = new TimeCoordinates();
pTransform1.SetWeekType (ChartObj.WEEK_5D);
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_NEAR,
                               ChartObj.AUTOAXES_NEAR);.
.
.       // Define axes, axes labels and grids
.
ChartAttribute attrib1 =
       new ChartAttribute(Color.Red, 1,ChartObj.DashStyle.Solid., Color.Red);
attrib1.SetFillFlag(true);
OHLCPlot thePlot1 = new OHLCPlot(pTransform1, Dataset1,
               ChartCalendar.GetCalendarWidthValue(ChartObj.DAY_OF_YEAR,0.75),
               attrib1);
chartVu.AddChartObject(thePlot1);
```

[Visual Basic]

```
Dim Dataset1 As New TimeGroupDataset("Stock Data", xValues, stockPriceData)
pTransform1 = New TimeCoordinates()
pTransform1.SetWeekType(weekmode)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_NEAR)
.
.       ' Define axes, axes labels and grids
.

Dim attrib1 As New ChartAttribute(Color.Red, 1, DashStyle.Solid, Color.Red)
attrib1.SetFillFlag(True)
 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.

Stacked Bar Plots

Class StackedBarPlot

GraphObj
|
+--ChartPlot
|
+--GroupPlot
|
+--StackedBarPlot

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.

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
);
```

transform	The coordinate system for the new **StackedBarPlot** object.
dataset	The stacked bar graph represents the values in this group dataset.
rbarwidth	The width of the stacked bars in units of the independent axis.
rbarbase	The stacked bars start at the value *rbarbase*, and extend to the group bar values represented by the dataset.
attribs	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.

nbarjust The stacked bars are justified with respect to the x-values in the dataset using the *rbarjust* 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 BigChartDemo, class GroupBarPlotChart)

[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
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)
pTransform2.SetTimeScaleStart(New ChartCalendar(1997, ChartObj.JANUARY, 1))
pTransform2.SetTimeScaleStop(New ChartCalendar(2003, ChartObj.JANUARY, 1))

pTransform2.SetGraphBorderDiagonal(0.55, 0.1, 0.95, 0.75)

pTransform2.SetScaleStartY(0)
.
.   ' Define axes, axes labels, and grids
.
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()
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
|
+--ChartPlot
|
+--GroupPlot
|

+--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.

StackedLinePlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal transform As PhysicalCoordinates, _
   ByVal dataset As GroupDataset, _
   ByVal attribs As ChartAttribute() _
)
[C#]
public StackedLinePlot(
   PhysicalCoordinates transform,
   GroupDataset dataset,
   ChartAttribute[] attribs
);
```

transform	The coordinate system for the new **StackedLinePlot** object.
dataset	The stacked line plot represents the values in this group dataset.
attribs	An array of **ChartAttribute** objects, sized the same as the number of groups in the dataset specify the attributes (line color) 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 cooresponds to the group number. These methods include SetSegmentAttributes, SetSegmentFillColor, SetSegmentLineColor, and SetSegmentColor.

Stacked line plot example (extracted from the example program BigChartDemo, class StackedLineChart)

[C#]

```
int  numPoints = 100;
int numGroups = 7;
double []x1 = new double[numPoints];
double [,]y1 = new double[numGroups,numPoints];
.
```

```
.  // Initialize data
.
theFont = new Font("Microsoft Sans Serif", 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]

```
Dim numPoints As Integer = 100
```

```
Dim numGroups As Integer = 7
Dim x1(numPoints-1) As Double
Dim y1(numGroups-1, numPoints-1) As Double
.
.  ' Initialize data
.
theFont = New Font("Microsoft Sans Serif", 10, FontStyle.Bold)
Dim Dataset1 As New GroupDataset("First", x1, y1)
Dataset1.SetStackMode(ChartObj.AUTOAXES_STACKED)
Dim pTransform1 As New CartesianCoordinates(ChartObj.LINEAR_SCALE, _
       ChartObj.LINEAR_SCALE)
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
pTransform1.SetScaleStartX(0)
pTransform1.SetScaleStartY(0)

Dim background As New Background(pTransform1, ChartObj.PLOT_BACKGROUND, _
       Color.FromArgb(255, 255, 255))
chartVu.AddChartObject(background)

pTransform1.SetGraphBorderDiagonal(0.15, 0.15, 0.9, 0.75)
.
. ' Define axes, axes labels and grids
.
Dim attrib1 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid)
attrib1.SetFillFlag(True)
attrib1.SetLineFlag(False)
Dim attribArray(numGroups) As ChartAttribute
For i = 0 To numGroups - 1
   attribArray(i) = attrib1.Clone()
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)

Dim thePlot1 As New StackedLinePlot(pTransform1, Dataset1, attribArray)
chartVu.AddChartObject(thePlot1)
```

12. Contour Plotting

ChartPlot
ContourPlot

The **ContourPlot** class displays contour data organized in a **ContourDataset**.dataset. The line contour graph draws contituous 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 contor 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 concrete implementation 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.

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
);
```

transform	The coordinate system for the new **ContourPlot** object.
dataset	The **ContourDataset** plot will represent the xyz values in this contour data set.
contourlevels	An array, size [numcontourlevels], of the contour levels used in the contour plot.
attribs	An array of color and fill attributes, size [numcontourlevels+1]. If the *contourtype* 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..*numcontourlevels* set the colors of the contour regions.
blineflags	An array, size [numcontourlevels], of boolean flags specifying whether a contour line should be displayed.
blabelflags	An array, size [numcontourlevels], of boolean flags specifying whether a contour line should be labeled with the numeric value of the associated contour level.
numcontourlevels	The number of contour levels.
contourtype	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 BigChartDemo, class ContourLinePlot)

[C#]

```
double []contourlevels = { 1000, 1200, 1400, 1600, 1800,
                              1900, 2000, 2100, 2200, 2400, 2600, 2800, 3000};
chartVu  = chartView1;
int i;
theFont = new Font("Microsoft Sans Serif", 10,  FontStyle.Bold);
chartVu  = chartView1;

CartesianCoordinates pTransform1 = new CartesianCoordinates(-7, -7, 7, 7);

CreateRegularGridPolysurface();
pTransform1.SetGraphBorderDiagonal(0.10, .10, .85, 0.85) ;
Background background =
       new Background( pTransform1, ChartObj.GRAPH_BACKGROUND, Color.White);
chartVu.AddChartObject(background);

ChartText checkBoxCaption =
   new ChartText(pTransform1, theFont,"Contour Level", 560, 35, ChartObj.DEV_POS);
chartVu.AddChartObject(checkBoxCaption);

LinearAxis xAxis = new LinearAxis(pTransform1, ChartObj.X_AXIS);
chartVu.AddChartObject(xAxis);
LinearAxis yAxis = new LinearAxis(pTransform1, ChartObj.Y_AXIS);
chartVu.AddChartObject(yAxis);
NumericAxisLabels xAxisLab = new NumericAxisLabels(xAxis );
chartVu.AddChartObject(xAxisLab);
NumericAxisLabels yAxisLab = new NumericAxisLabels(yAxis);
chartVu.AddChartObject(yAxisLab);
Grid xgrid = new Grid(xAxis, yAxis,ChartObj.X_AXIS, ChartObj.GRID_MAJOR);
chartVu.AddChartObject(xgrid);
```

```
Grid ygrid = new Grid(xAxis, yAxis,ChartObj.Y_AXIS, ChartObj.GRID_MAJOR);
chartVu.AddChartObject(ygrid);

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,2,DashStyle.Solid,color);
     attribs[i].SetFillFlag(true);
}
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);

for (i=0; i < numcontourlevels; i++)
{
   if ((i % 3) == 0)
        lineflags[i] = true;
    else
        lineflags[i] = false;
    if ((i % 3) == 0)
        labelflags[i] = true;
    else
        labelflags[i] = false;
}

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() = _
    {1000, 1200, 1400, 1600, 1800, 1900, 2000, 2100, 2200, 2400, 2600, 2800, 3000}
Dim theFont As Font
chartVu = ChartView1
Dim i As Integer
theFont = New Font("Microsoft Sans Serif", 10, FontStyle.Bold)
Dim pTransform1 As New CartesianCoordinates(-7, -7, 7, 7)

CreateRegularGridPolysurface()
'   CreateRandomGridPolysurface()
pTransform1.SetGraphBorderDiagonal(0.1, 0.1, 0.85, 0.85)
Dim background As New Background(pTransform1, ChartObj.GRAPH_BACKGROUND, _
     color.White)
chartVu.AddChartObject(background)

Dim checkBoxCaption As New ChartText(pTransform1, theFont, "Contour Level", _
       560, 35, ChartObj.DEV_POS)
chartVu.AddChartObject(checkBoxCaption)

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)
chartVu.AddChartObject(xAxisLab)

Dim yAxisLab As New NumericAxisLabels(yAxis)
chartVu.AddChartObject(yAxisLab)

Dim xgrid As New Grid(xAxis, yAxis, ChartObj.X_AXIS, ChartObj.GRID_MAJOR)
chartVu.AddChartObject(xgrid)

Dim ygrid As New Grid(xAxis, yAxis, ChartObj.Y_AXIS, ChartObj.GRID_MAJOR)
```

```
chartVu.AddChartObject(ygrid)

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)

For i = 0 To numcontourlevels - 1
  If i Mod 3 = 0 Then
    lineflags(i) = True
  Else
    lineflags(i) = False
  End If
  If i Mod 3 = 0 Then
     labelflags(i) = True
  Else
    labelflags(i) = False
   End If
Next i
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 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 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.

Marker constructor

```
[Visual Basic]
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 rsize,
   int npostype
);
```

transform	Places the marker in the coordinate system defined by transform.

nmarkertype	Specifies the shape of the current chart marker. Use one of the chart marker constants: MARKER_NULL, MARKER_VLINE, MARKER_HLINE, MARKER_CROSS, MARKER_BOX or MARKER_HVLINE.
x	Specifies the x-value of the marker position
y	Specifies the y-value of the marker position
rsize	Specifies the size of the cross hair marker (MARKER_CROSS) and the box marker (MARKER_BOX) in .Net CF device coordinates.
npostype	Specifies the if the position of the marker is specified in physical coordinates, normalized coordinates or .Net CF device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.

The marker constants signify:

MARKER_NULL	An invisible marker
MARKER_VLINE	The marker is a vertical line extending from the top of the plot area 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 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 CF 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 CF 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.

Simple marker example

[C#]

```
CartesianCoordinates pTransform1 =
       new CartesianCoordinates( 0.0, 0.0, 10.0, 20.0);
pTransform1.AutoScale(Dataset1,  ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR);
.
.
.
double xpos = 5.0;
double ypos = 10.0;
Marker amarker = new Marker(pTransform1, ChartObj.MARKER_BOX, xpos, ypos,
                  10, ChartObj.PHYS_POS);
theChartView.AddChartObject(amarker);
```

[Visual Basic]

```
Dim pTransform1 As CartesianCoordinates =  _
       New CartesianCoordinates( 0.0, 0.0, 10.0, 20.0)
pTransform1.AutoScale(Dataset1,  ChartObj.AUTOAXES_FAR, ChartObj.AUTOAXES_FAR)
.
.
.
Dim xpos As Double = 5.0
Dim ypos As Double = 10.0
Dim amarker As Marker = New Marker(pTransform1, ChartObj.MARKER_BOX, _
           xpos, ypos,  10, ChartObj.PHYS_POS)
theChartView.AddChartObject(amarker)
```

Data Cursors

Class DataCursor

GraphObj
|
+--Marker
|
+-- DataCursor

Data cursors are an extension of the marker class. Data cursors combine the .Net CF 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.

DataCursor constructor

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal component As ChartView, _
   ByVal transform As PhysicalCoordinates, _
   ByVal nmarkertype As Integer, _
   ByVal rsize As Double _
)

[C#]
public DataCursor(
   ChartView component,
   PhysicalCoordinates transform,
   int nmarkertype,
   double rsize
);
```

component	A reference to the **ChartView** object that the chart is placed in.
transform	The **PhysicalCoordinates** object associated with the data cursor.
nmarkdertype	The marker type. Use one of the **Marker** marker type constants: MARKER_VLINE .. MARKER_ BOX.
rsize	The size in .Net CF 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 (Adapted from the DataCursorView and CustomChartDataCursor classes)

[C#]

```
public class CustomChartDataCursor extends DataCursor {
// Create your own custom constructor and call the parent constructor

public CustomChartDataCursor(ChartView achartview,
             CartesianCoordinates thetransform,
             int nmarkertype,
              double rsize): base(achartview, thetransform, nmarkertype, rsize)
  {

  }

 // The mouse Released event should look like this
  public void OnMouseUp (MouseEvent event)
  {
     base.MouseReleased(mouseevent);
     // Add your own code here
  }
}
.
.
```

```
.

CustomChartDataCursor dataCursorObj =
    new CustomChartDataCursor( chartVu, pTransform1, ChartObj.MARKER_HVLINE, 8.0);
dataCursorObj.SetEnable(true);
chartVu.SetCurrentMouseListener(dataCursorObj);
```

[Visual Basic]

```
Public Class CustomChartDataCursor Inherits DataCursor
   Public Sub New(ByVal achartview As ChartView, _
           ByVal thetransform As CartesianCoordinates, _
           ByVal nmarkertype As Integer, ByVal rsize As Double)
          MyBase.New(achartview, thetransform, nmarkertype, rsize)
   End Sub 'New


   Public Overrides Sub OnMouseUp(ByVal mouseevent As MouseEventArgs)
      MyBase.OnMouseUp(mouseevent)
      ' Add your own code here

   End Sub 'OnMouseUp
End Class 'CustomChartDataCursor
.
.
.
Dim dataCursorObj As New _
       CustomChartDataCursor(chartVu, pTransform1,  ChartObj.MARKER_HVLINE, 8.0)
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 below extends the previous **Marker** and **DataCursor** examples. In this 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 DataCursorView and CustomChartDataCursor classes)

[C#]

```
public class CustomChartDataCursor: DataCursor
{
    NumericLabel pointLabel;
    Font textCoordsFont = new Font("Microsoft Sans Serif", 8, FontStyle.Regular);
    double rNumericLabelCntr = 0.0;
    SimpleLinePlot thePlot1;
    SimpleLinePlot thePlot2;

    public CustomChartDataCursor(ChartView achartview,
        CartesianCoordinates thetransform,
        SimpleLinePlot plot1,
        SimpleLinePlot plot2,
        int nmarkertype,
        double rsize): base(achartview, thetransform, nmarkertype, rsize)
    {
        thePlot1 = plot1;
        thePlot2 = plot2;
    }

    public override void MouseReleased (MouseEventArgs mouseevent)
    {
        NearestPointData nearestPointObj1 =  new NearestPointData();
        NearestPointData nearestPointObj2 =  new NearestPointData();
        Point2D nearestPoint = new Point2D(0,0);
        ChartView chartVu = GetChartObjComponent();
        bool bfound1 = false;
```

```
        bool bfound2 = false;

        base.MouseReleased(mouseevent);
        if ((mouseevent.Button & GetButtonMask()) != 0)
        {

            // Find nearest point for each line plot object
            Point2D location = GetLocation();
            bfound1 = thePlot1.CalcNearestPoint(location,
                    ChartObj.FNP_NORMDIST, nearestPointObj1);
            bfound2 = thePlot2.CalcNearestPoint(location,
                    ChartObj.FNP_NORMDIST, nearestPointObj2);

            if (bfound1 && bfound2)
            {
            // choose the nearest point
                if (nearestPointObj1.GetNearestPointMinDistance() <
                        nearestPointObj2.GetNearestPointMinDistance())
                    nearestPoint =  nearestPointObj1.GetNearestPoint();
                else
                    nearestPoint = nearestPointObj2.GetNearestPoint();
            // create marker object at place it at the nearest point
                Marker amarker =
                        new Marker(GetChartObjScale(), MARKER_BOX,
                        nearestPoint.GetX(),   nearestPoint.GetY(),
                        10.0, PHYS_POS);
                chartVu.AddChartObject(amarker);
                rNumericLabelCntr += 1.0;
            // Add a numeric label the identifies the marker
                pointLabel =
                        new NumericLabel(GetChartObjScale(),
                        textCoordsFont, rNumericLabelCntr,
                        nearestPoint.GetX(), nearestPoint.GetY(),
                        PHYS_POS, DECIMALFORMAT, 0);
        // Nudge text to the right and up so that it does not write over marker
                pointLabel.SetTextNudge(5,-5);
                chartVu.AddChartObject(pointLabel);
                chartVu.UpdateDraw();
                }
            }
        }
    }
```

[Visual Basic]

```
Public Class CustomChartDataCursor
   Inherits DataCursor
   Private pointLabel As NumericLabel
   Private textCoordsFont As New Font("Microsoft Sans Serif", 8,
FontStyle.Regular)
   Private rNumericLabelCntr As Double = 0.0
   Private cplot1 As SimpleLinePlot
   Private cplot2 As SimpleLinePlot

   Public Sub New(ByVal achartview As ChartView, _
                  ByVal thetransform As CartesianCoordinates, _
                  ByVal plot1 As SimpleLinePlot, _
                  ByVal plot2 As SimpleLinePlot, _
                  ByVal nmarkertype As Integer, _
                  ByVal rsize As Double)
      MyBase.New(achartview, thetransform, nmarkertype, rsize)
      cplot1 = plot1
      cplot2 = plot2
   End Sub 'New


    Public Overrides Sub OnMouseUp(ByVal mouseevent As MouseEventArgs)
       Dim nearestPointObj1 As New NearestPointData()
       Dim nearestPointObj2 As New NearestPointData()
       Dim nearestPoint As New Point2D(0, 0)
       Dim chartview As ChartView = GetChartObjComponent()
       Dim bfound1 As Boolean = False
       Dim bfound2 As Boolean = False

       MyBase.OnMouseUp(mouseevent)

       If (mouseevent.Button And GetButtonMask()) <> 0 Then
          ' Find nearest point for each line plot object
          Dim location As Point2D = GetLocation()
          bfound1 = cplot1.CalcNearestPoint(location, _
               ChartObj.FNP_NORMDIST, nearestPointObj1)
          bfound2 = cplot2.CalcNearestPoint(location, _
               ChartObj.FNP_NORMDIST, nearestPointObj2)
          If bfound1 And bfound2 Then
                    ' choose the nearest point
          If nearestPointObj1.GetNearestPointMinDistance() < _
             nearestPointObj2.GetNearestPointMinDistance() Then
```

```
                nearestPoint = nearestPointObj1.GetNearestPoint()
            Else
               nearestPoint = nearestPointObj2.GetNearestPoint()
            End If
    ' create marker object at place it at the nearest point
       Dim amarker As New Marker(GetChartObjScale(), MARKER_BOX, _
               nearestPoint.GetX(), nearestPoint.GetY(), 10.0, PHYS_POS)
       chartview.AddChartObject(amarker)
       rNumericLabelCntr += 1.0
     ' Add a numeric label the identifies the marker
         pointLabel = New NumericLabel(GetChartObjScale(), textCoordsFont, _
                  rNumericLabelCntr, nearestPoint.GetX(), nearestPoint.GetY(), _
                  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()
      End If
    End If
  End Sub 'OnMouseUp
End Class 'CustomChartDataCursor
```

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 OHLCFinPlot demonstrates how to use a **ChartText** object as a popup box to display this type of data. In another related example, the program **LabeledPieChart** shows how to trap a click on a specific pie slice and display additional data for that slice using a **ChartText** object.

14. Moving Chart Objects, Data Points and Coordinate Systems

MoveObj
MoveData
MoveCoordinates

Many of the subclasses of **GraphObj** are moveable using the mouse. This includes the axis, legend, text, image, 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.

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.

Starting in Revision 2.0 we have added the capability of moving the coordinates system of a graph, using the **MoveCoordinates** class. The move operation is analogous to the way you can change the latitude and longitude of an internet map by clicking and dragging it.

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.

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 MouseButtons, _
```

```
   ByVal object1filter As String _
)


Overloads Public Sub New( _
   ByVal component As ChartView, _
   ByVal buttonmask As MouseButtons _
)


[C#]
public MoveObj(
   ChartView component,
   MouseButtons buttonmask,
   string object1filter
);

public MoveObj(
   ChartView component,
   MouseButtons buttonmask
);
```

component	A reference to the **ChartView** object that the chart is placed in.
buttonmask	Specifies the mouse button that is trapped to invoke a move.
objectfilter	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. Attemp 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 (Adapted from the MoveObjects class)

[C#]

```
ChartView chartVu  = chartView1;
CartesianCoordinates pTransform1 =
       new CartesianCoordinates( 0.0, 0.0, 10.0, 20.0);
pTransform1.AutoScale(Dataset1,  ChartObj.AUTOAXES_FAR,
                     ChartObj.AUTOAXES_FAR);
.
.
.
MoveObj mouselistener = new MoveObj(chartVu );
mouselistener.SetEnable(true);
mouselistener.SetMoveObjectFilter("GraphObj");
chartVu.SetCurrentMouseListener(mouselistener);
```

[Visual Basic]

```
Dim chartVu As ChartView = ChartView1
Dim pTransform1 As CartesianCoordinates =
       New CartesianCoordinates( 0.0, 0.0, 10.0, 20.0)
pTransform1.AutoScale(Dataset1,  ChartObj.AUTOAXES_FAR, _
                     ChartObj.AUTOAXES_FAR)
.
.
.
Dim mouselistener As MoveObj = New MoveObj(chartVu )
mouselistener.SetEnable(True)
mouselistener.SetMoveObjectFilter("GraphObj")
chartVu.SetCurrentMouseListener(mouselistener)
```

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 moue buttons 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.

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
);
```

component	A reference to the **ChartView** object that the chart is placed in.
transform	The **PhysicalCoordinates** object associated with the **MoveData** object.
buttonmask	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 (See the class MoveDatapoint for a more complicated example)

[C#]

```
ChartView chartVu  = chartView1;
CartesianCoordinates pTransform1 =
       new CartesianCoordinates( 0.0, 0.0, 10.0, 20.0);
pTransform1.AutoScale(Dataset1,  ChartObj.AUTOAXES_FAR,
                     ChartObj.AUTOAXES_FAR);
.
.
MoveData mouselistener = new MoveData (chartVu, pTransform1 );
mouselistener.SetMarkerType( ChartObj.MARKER_CROSS);
mouselistener.SetMarkerSize(12);
mouselistener.SetMoveMode(ChartObj.MOVE_Y);
mouselistener.SetEnable(true);
chartVu.SetCurrentMouseListener(mouselistener);
```

[Visual Basic]

```
Dim chartVu As ChartView = ChartView1
Dim pTransform1 As CartesianCoordinates = _
       New CartesianCoordinates( 0.0, 0.0, 10.0, 20.0)
pTransform1.AutoScale(Dataset1,  ChartObj.AUTOAXES_FAR, _
                     ChartObj.AUTOAXES_FAR)
.
.
.
Dim mouselistener As MoveData = New MoveData (chartVu, pTransform1 )
mouselistener.SetMarkerType( ChartObj.MARKER_CROSS)
```

```
mouselistener.SetMarkerSize(12)
mouselistener.SetMoveMode(ChartObj.MOVE_Y)
mouselistener.SetEnable(True)
chartVu.SetCurrentMouseListener(mouselistener)
```

Moving the Chart Coordinate System

Class MoveCoordinates

MouseListener
|
+--DataCursor
|
+--MoveCoordinates

The **MoveCoordinates** mouse listener traps a mouse pressed event. While the mouse is button is held down, the underlying coordinate system will track the move movements. Release the mouse button and the chart redraws one final time using the extents of the final coordinate system.

MoveCoordinates constructors

```
Visual Basic (Declaration)
Public Sub New ( _
    component As ChartView, _
    transform As PhysicalCoordinates, _
    buttonmask As MouseButtons _
)
C#
public MoveCoordinates(
    ChartView component,
    PhysicalCoordinates transform,
    MouseButtons buttonmask
)
```

component	A reference to the **ChartView** object that the chart is placed in.
transform	The coordinate system underlying the chart.
buttonmask	Specifies the mouse button that is trapped to invoke a move.

Create the **MoveCoordinates** object and then install it using the **ChartView.SetCurrentMouseListener** method. This adds the **MoveCoordinates** object as a **MouseListener** to the **ChartView** object. Enable/Disable the function using MoveCoordinates**.SetEnable** method. Call MoveCoordinates**.SetCurrentMouseListener**(null) to remove the object as a mouse listener for the chart view.

You can restrict the movement of the coordinate system to the x-dimension (`MOVE_X`), or the y-dimension (`MOVE_Y`), using the **MoveMode** property. Set **MoveMode** to MOME_XY for unrestricted movement.

If you have multiple coordinate systems in the chart, you can move them all simultaneously by setting the **MultiTransformMove** property true. Otherwise, only the coordinate system passed into the constructor is updated with the move information.

Moving the coordinate system (Extracted from the ZoomExamples.MoveCoordinates example)

[C#]

```
TimeSimpleDataset Dataset1 = new TimeSimpleDataset("First", tradingDay,
stockPrice);


TimeCoordinates pTransform1 = new TimeCoordinates();
pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_FAR);.
.
.
.
MoveCoordinates movecoords = new MoveCoordinates(chartVu, pTransform1);
movecoords.SetEnable(true);
chartVu.SetCurrentMouseListener(movecoords);
```

[Visual Basic]

```
Dim Dataset1 As New TimeSimpleDataset("First", timeData, dataValues)
Dim pTransform1 As New TimeCoordinates()


pTransform1.AutoScale(Dataset1, ChartObj.AUTOAXES_NEAR, ChartObj.AUTOAXES_FAR)
.
.
```

```
.
Dim movecoords As New MoveCoordinates(chartVu, pTransform1)
movecoords.SetEnable(True)
chartVu.SetCurrentMouseListener(movecoords)
```

15. Zooming

ChartZoom

Zooming is the interactive re-scaling of a charts 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.

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.
- The zoomed coordinate system can be forced to maintain a fixed aspect ratio.
- User-defineable zoom limits prevent numeric under and overflows

Magnification is related to zooming because it is also the interactive re-scaling of a charts physical coordinate system. In this case, a fixed sized view window is passed over a source chart, analogous to passing a magnifying class over a map. The area under the view window is “magnified” and redisplayed in a separate target chart. The source chart does not re-scale, as in the case of zooming. Only the target chart gets rescaled, in response to the position of the view window position over the source chart.

Simple Zooming of a single physical coordinate system

Class ChartZoom

MouseListener

```
|
+--ChartZoom
```

The **ChartZoom** class implements .Net CF delegates for mouse events. It implements and uses the mouse events: OnMouseMove, OnDoubleClick, OnMouseDown, OnMouseUp and OnClick. 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 mouse released event. During the mouse released 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 for 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 _
)
[C#]
public ChartZoom(
   ChartView component,
   PhysicalCoordinates transform,
   bool brescale
);
```

component	A reference to the **ChartView** object that the chart is placed in.
transform	The **PhysicalCoordinates** object associated with the scale being zoomed.
brescale	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 is to its original condition, after which the PopZoomStack method has no effect.

Integrated zoom stack processing
Starting with Revision 2.0, zoom stack processing is internal to **ChartZoom** class. There is no need to subclass the **ChartZoom** class in order to implement a zoom stack. Just set the ChartZoom.InternalZoomStackProcessing property true.

```
zoomObj.InternalZoomStackProcesssing = true;
```

Return to a previous zoom level by right clicking the mouse. Change the zoom stack button using the ZoomStackButtonMask property. Setting it to MouseButtons.Left, MouseButtons.Right or MouseButtons.Middle.

Aspect Ratio Correction
Starting with Revision 2.0, you can force the zoom rectangle to maintain a fixed aspect ratio. Use the ChartZoom.ArCorrectionMode property to specify the aspect ratio correction mode.

ZOOM_NO_AR_CORRECTION	Allow the x- and y-dimension of the zoom rectangle to change the overall charts physical aspect ratio. This is the default mode, and the only mode supported prior to Revision 2.0.
ZOOM_X_AR_CORRECTION	Track the x-dimension of the zoom rectangle and calculate the y-dimension in order to maintain a fixed aspect ratio.
ZOOM_Y_AR_CORRECTION	Track the y-dimension of the zoom rectangle and calculate the x-dimension in order to maintain a fixed aspect ratio.

The target aspect ratio is the aspect ratio of the coordinate system(s) at the time the **ChartZoom** object is intitialized.

```
zoomObj.ArCorrectionMode = ChartObj.ZOOM_X_AR_CORRECTION
```

Simple zoom example (Adapted from the SimpleZoom example)
In this example, a new class derives from the **ChartZoom** class and the MousePressed event overridden. The event invokes the PopZoomStack method. Otherwise, the default operation of the **ChartZoom** class controls everything else.

[C#]

```
private class ZoomWithStack: ChartZoom
{
    public ZoomWithStack(ChartView component,
                  CartesianCoordinates transform, bool brescale):
        base( component,  transform, brescale)
        {
        }
        public override void MousePressed (MouseEventArgs mouseevent)
        {
            if ((mouseevent.Button & MouseButtons.Right) != 0)
                this.PopZoomStack();
            else
                base.MousePressed(mouseevent);
        }
    }
.
.
.
ZoomWithStack zoomObj = new ZoomWithStack(chartVu, pTransform1, 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 Dimension(1.0, 1.0));
chartVu.SetCurrentMouseListener(zoomObj);
```

[Visual Basic]

```
Private Class ZoomWithStack
        Inherits ChartZoom
```

```
    Public Sub New(ByVal component As ChartView, _
            ByVal transform As CartesianCoordinates, ByVal brescale As Boolean)
        MyBase.New(component, transform, brescale)
    End Sub 'New

    Public Overrides Sub OnMouseDown(ByVal mouseevent As MouseEventArgs)
      If (mouseevent.Button And MouseButtons.Right) <> 0 Then
        Me.PopZoomStack()
      Else
        MyBase.OnMouseDown(mouseevent)
     End If
    End Sub 'OnMouseDown
End Class 'ZoomWithStack
.
.
.
Dim zoomObj As New ZoomWithStack(chartVu, pTransform1, 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 Dimension(1.0, 1.0))
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 mouse released 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 _
)
[C#]
public ChartZoom(
   ChartView component,
   PhysicalCoordinates[] transforms,
   bool brescale
);
```

component	A reference to the **ChartView** object that the chart is placed in.
transforms	An array, size numtransforms, of the **PhysicalCoordinates** objects associated with the zoom operation.
brescale	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.

Starting with Revision 2.0, zoom stack processing is internal to **ChartZoom** class. There is no need to subclass the **ChartZoom** class in order to implement a zoom stack. Just set the ChartZoom.InternalZoomStackProcessing property true.

```
zoomObj.InternalZoomStackProcesssing = true;
```

Return to a previous zoom level by right clicking the mouse. Change the zoom stack button using the ZoomStackButtonMask property. Setting it to MouseButtons.Left, MouseButtons.Right or MouseButtons.Middle.

Super zoom example (Adapted from the SuperZoom example)
In this example, a new class derives from the **ChartZoom** class and the MousePressed event overridden. The event invokes the PopZoomStack method. Otherwise, the default operation of the **ChartZoom** class controls everything else.

[C#]

```
public class SuperZoom : com.quinncurtis.chart2dnetcf.ChartView
{
    private class ZoomWithStack: ChartZoom
    {
        public ZoomWithStack(ChartView component,
                CartesianCoordinates []transforms,  bool brescale):
                base( component,  transforms, brescale)
            {
            }
        public override void MousePressed (MouseEventArgs mouseevent)
        {
            if ((mouseevent.Button & MouseButtons.Right) != 0)
                this.PopZoomStack();
            else
                base.MousePressed(mouseevent);
        }
    }
.
.
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);
.
.

CartesianCoordinates []transformArray =
        {pTransform1, pTransform2,pTransform3,pTransform4,pTransform5};

ZoomWithStack zoomObj = new ZoomWithStack(chartVu, transformArray, 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 MouseButtons.Right) <> 0 Then
         Me.PopZoomStack()
      Else
         MyBase.OnMouseDown(mouseevent)
      End If
    End Sub 'OnMouseDown
 End Class 'ZoomWithStack
.
.
.
 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)

 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)
.
.
.
Dim transformArray As CartesianCoordinates() = {pTransform1, _
           pTransform2, pTransform3, pTransform4, pTransform5}

Dim zoomObj As New ZoomWithStack(chartVu, transformArray, 5, 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 infinte zooming can result in numeric under and overflows. The default minimum allowable range resuling 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)
```

Magnifying a portion of a chart in a separate window

Class MagniView

MouseListener

|

+--MagniView

The **MagniView** class needs two chart areas in order to work: a source chart area, which contains the full scale display of the chart, and the target chart area, which will contain a magnified view of the chart.

The **MagniView** class starts the magnify operation on the OnMouseDown event, positioning the lower left corner of a magnify rectangle on top of the source chart. Immediately, a magnified view of the chart will appear in the target chart. As the mouse moves the magnify rectangle, the target chart constantly updates to reflect the current view window. Once the mouse button is released, the target chart remains at its current values.

MagniView constructor

The constructor below creates a **MagniView** object for a single chart coordinate system.

```
Visual Basic (Declaration)
Public Sub New ( _
    source As ChartView, _
    target As ChartView, _
    transform As PhysicalCoordinates, _
    magnirect As Dimension, _
)
C#
public MagniView(
    ChartView source,
    ChartView target,
    PhysicalCoordinates transform,
    Dimension magnirect
)
```

source	A reference to the source **ChartView** object that the chart is placed in.
target	A reference to the target **ChartView** object that the magnified view of the chart is placed in.
transform	The source **PhysicalCoordinates** object associated with the scale being magnified.
magnirect	The rectangle, in physical coordinates, of the magnify cursor.

Enable the magnify object after creation using the **MagniView.SetEnable(true)** method.

Retrieve the physical coordinates of the magnify rectangle using the **MagniView** GetMagniMin and GetMagniMax methods. Restrict magnification in the x- or y-direction using the SetMagniXEnable and SetMagniYEnable methods. Set the rounding mode associated with rescale operations using the SetMagniXRoundMode and SetMagniYRoundMode methods.

In order to use the **MagniView** class, you need two instances of the underlying chart. Place one above, or next to the other, on a parent form, as done in the example. The first instance of the chart should be considered the source chart, and the second instance of the chart should be considered the target. This way you end up with two charts with an identical set of axes and plot ojects.

Simple magnify example (Adapted from the ZoomExamples.MagniViewUserControl1 example)

[C#]

```
InitializeData();


// Define charts using data
magniViewChart1.InitializeChart(Dataset1, Dataset2, "Click and drag on the top
graph using left mouse button.","", false);
magniViewChart2.InitializeChart(Dataset1, Dataset2, "The area bounded by the mouse
'magnify' icon is displayed full-scale in the bottom graph.", "", true);


// Hook-up the MagniView class to the source and target classes
MagniView magnifyObj = new MagniView(magniViewChart1,
            magniViewChart2, magniViewChart1.pTransform1,
            new Dimension(0.05, 0.2));
            magnifyObj.SetButtonMask(MouseButtons.Left);
            magnifyObj.SetEnable(true);
            magnifyObj.UpdateDuringDrag = true;
            magniViewChart1.SetCurrentMouseListener(magnifyObj);
```

[Visual Basic]

```
InitializeData()


' Define charts using data
MagniViewChart1.InitializeChart(Dataset1, Dataset2, _
    "Click and drag on the top graph using left mouse button.", "", False)
MagniViewChart2.InitializeChart(Dataset1, Dataset2, _
    "The area bounded by the mouse 'magnify' icon is displayed full-scale in the
bottom graph.", "", True)
'
' Hook-up the MagniView class to the source and target classes
Dim magnifyObj As New MagniView(MagniViewChart1, MagniViewChart2, _
```

```
    MagniViewChart1.pTransform1, New Dimension(0.05, 0.2))
magnifyObj.SetButtonMask(MouseButtons.Left)
magnifyObj.SetEnable(True)
magnifyObj.UpdateDuringDrag = True
MagniViewChart1.SetCurrentMouseListener(magnifyObj)
```

Magnifying multiple physical coordinate systems

The **MagniView** class also supports the magnifications of multiple physical coordinate systems. If the source chart has multiple coordinate systems, and you want to magnify all of the coordinate systems, use the **MagniView** constructor which takes an array of **PhysicalCoordinates** systems as an argument.

MagniView constructor

Visual Basic (Declaration)

```
Public Sub New ( _
    source As ChartView, _
    target As ChartView, _
    transforms As PhysicalCoordinates(), _
    magnirect As Dimension _
)
```

C#

```
public MagniView(
    ChartView source,
    ChartView target,
    PhysicalCoordinates[] transforms,
    Dimension magnirect
)
```

source	A reference to the source **ChartView** object that the chart is placed in.
target	A reference to the target **ChartView** object that the magnified view of the chart is placed in.
transforms	An arry of **PhysicalCoordinates** objects associated with the chart being magnified.
magnirect	The rectangle, in physical coordinates, of the magnify cursor.

Enable the magnify object after creation using the **MagniView.SetEnable(true)** method.

Retrieve the physical coordinates of the magnify rectangle using the **MagniView** GetMagniMin and GetMagniMax methods. Restrict magnification in the x- or y-direction using the SetMagniXEnable and SetMagniYEnable methods. Set the rounding mode associated with rescale operations using the SetMagniXRoundMode and SetMagniYRoundMode methods.

In order to use the **MagniView** class, you need two instances of the underlying chart. Place one above, or next to the other, on a parent form, as done in the example. The first instance of the chart should be considered the source chart, and the second instance of the chart should be considered the target. This way you end up with two charts with an identical set of axes and plot ojects.

Super magnify example (Adapted from the ZoomExamples.SuperMagniViewUserControl1 example)

[C#]

```
InitializeData();


// Define charts using data
superMagniViewChart1.InitializeChart(Dataset1, Dataset2, Dataset3, Dataset4,
Dataset5,
"Click and drag on the top graph using left mouse button.", "");
superMagniViewChart2.InitializeChart(Dataset1, Dataset2, Dataset3, Dataset4,
Dataset5, "The area bounded by the mouse 'magnify' icon is displayed full-scale in
the bottom graph.", "");


// Hook-up the MagniView class to the source and target classes
CartesianCoordinates[] transformArray = { superMagniViewChart1.pTransform1,
superMagniViewChart1.pTransform2, superMagniViewChart1.pTransform3,
superMagniViewChart1.pTransform4,  superMagniViewChart1.pTransform5 };


MagniView magnifyObj = new MagniView(superMagniViewChart1, superMagniViewChart2,
transformArray, new Dimension(0.1, 0.2));


magnifyObj.SetButtonMask(MouseButtons.Left);
magnifyObj.SetEnable(true);
superMagniViewChart1.SetCurrentMouseListener(magnifyObj);
```

[VB]

```
InitializeData()


' Define charts using data
```

```
SuperMagniViewChart1.InitializeChart(Dataset1, Dataset2, Dataset3, _
    Dataset4, Dataset5, "Click and drag on the top graph using left mouse
button.",  "")
SuperMagniViewChart2.InitializeChart(Dataset1, Dataset2, Dataset3, _
    Dataset4, Dataset5, "The area bounded by the mouse 'magnify' icon is displayed
full-scale in the bottom graph.", "")


' Hook-up the MagniView class to the source and target classes
Dim transformArray As CartesianCoordinates() ={SuperMagniViewChart1.pTransform1, _
    SuperMagniViewChart1.pTransform2, SuperMagniViewChart1.pTransform3, _
    SuperMagniViewChart1.pTransform4, SuperMagniViewChart1.pTransform5}
'
Dim magnifyObj As New MagniView(SuperMagniViewChart1, SuperMagniViewChart2, _
    transformArray, New Dimension(0.1, 0.2))


magnifyObj.SetButtonMask(MouseButtons.Left)
magnifyObj.SetEnable(True)
SuperMagniViewChart1.SetCurrentMouseListener(magnifyObj)
```

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


The **DataToolTip** class implements .Net CF mouse event delegates. It implements and uses the mouse events: OnMouseMove, OnDoubleClick, OnMouseDown, OnMouseUp, and OnClick. 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
);
```

component	A reference to the **ChartView** object that the chart is placed in.
buttonmask	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 (Adapted from the MultiAxes example)
In this example, the tooltip will display the decimal y-value of the nearest data point when the left mouse button is pressed.

[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 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("Microsoft Sans Serif", 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("Microsoft Sans Serif", 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 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.

[C#]

```
Font toolTipFont = new Font("Microsoft Sans Serif", 10, FontStyle.Regular);
DataToolTip datatooltip = new DataToolTip(chartVu);
TimeLabel xValueTemplate = new TimeLabel( ChartObj.TIMEDATEFORMAT_MDY);
NumericLabel yValueTemplate = new NumericLabel(ChartObj.CURRENCYFORMAT,2);
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("Microsoft Sans Serif", 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.

- Subclass the **DataToolTip** class with one of your own.
- Enable the custom mode by calling **DataToolTip.SetDataToolTipFormat**(ChartObj.DATA_TOOLTIP_CUSTOM) method.
- Override the **OnMouseDown** and **OnMouseUp** events and add the code needed to customize the display. In your custom **OnMouseDown** event, make sure you call **super**.**OnMouseDown** first, since this selects the plot object for you, and makes the plot objects coordinate system, dataset, and selected data point available using get methods. You can place your tooltip text in the **ChartText** object internal to the **DataToolTip** class. Get a reference to this object using the **DataToolTip.GetTextTemplate**() method. In your custom **OnMouseUp** event call **super.OnMouseUp** followed by a call to the **ChartView** repaint method (chartVu.UpdateDraw() in the example below);

Custom DataToolTip example (Adapted from the OHLCFinPlot example)
In this example, a new class is derived from the **DataToolTip** class and the OnMouseDown and OnMouseUp events are overridden.

[C#]

```
ChartView chartVu;
class CustomToolTip: DataToolTip
{
```

```
    ChartText stockpanel;
    ChartCalendar []xValues;
    double [,]stockPriceData;
    double []stockVolumeData;
    double []NASDAQData ;

    public CustomToolTip(ChartView component,
        ChartCalendar []xvalues, double [,]stockpricedata,
        double []nasdaqdata, double []stockvolumedata): base (component)
        {
            xValues = xvalues;
            stockPriceData = stockpricedata;
            stockVolumeData = stockvolumedata;
            NASDAQData = nasdaqdata;
            stockpanel = GetTextTemplate();
        }

        public override void  OnMouseUp(MouseEventArgs mouseevent)
        {
            base.OnMouseUp(mouseevent);
// Redraws the chart. Since the stockpanel object has not been added to the chart
// (using addChartObject) it will not be redrawn when the chart is redrawn
            GetChartObjComponent().UpdateDraw();
        }

        public override void  OnMouseDown (MouseEventArgs mouseevent)
        {
            Point2D mousepos = new Point2D();
            mousepos.SetLocation(mouseevent.X, mouseevent.Y);
            base.OnMouseDown(mouseevent);

            ChartPlot selectedPlot =  (ChartPlot) GetSelectedPlotObj();
            if (selectedPlot != null)
            {
                int selectedindex = GetNearestPoint().nearestPointIndex;
                PhysicalCoordinates transform = GetSelectedCoordinateSystem();
                stockpanel.SetChartObjScale(transform);
                stockpanel.SetLocation( mousepos, ChartObj.DEV_POS);
                stockpanel.SetTextString("Stock Data" );
    // 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 = stockPriceData[0,selectedindex];
```

```
    double high = stockPriceData[1,selectedindex];
    double low = stockPriceData[2,selectedindex];
    double close = stockPriceData[3,selectedindex];
double nasdaq = NASDAQData[selectedindex];
double volume = 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,
    xValues[selectedindex], ChartObj.TIMEDATEFORMAT_STANDARD);

stockpanel.AddNewLineTextString(timelabel.GetTextString());
stockpanel.AddNewLineTextString("Open " + openObj);
stockpanel.AddNewLineTextString("High " + highObj);
stockpanel.AddNewLineTextString("Low " + lowObj);
stockpanel.AddNewLineTextString("Close " + closeObj);
stockpanel.AddNewLineTextString("Volume " + volumeObj);
stockpanel.AddNewLineTextString("NASDAQ " + nasdaqObj);
stockpanel.SetChartObjEnable(ChartObj.OBJECT_ENABLE);
Graphics g2 = GetToolTipGraphics();
// Precalculates the text bounding box so that the size is
// known before it is drawn
stockpanel.PreCalcTextBoundingBox(g2);
Rectangle2D boundingbox = stockpanel.GetTextBox();
// Reposition tooltip text box if top of box near top of graph window
// You can do the same thing for all four sides of the graph window
if (  (mousepos.GetY() - boundingbox.GetHeight()) < 1)
{
    mousepos.SetLocation(mousepos.GetX(),
                mousepos.GetY() + boundingbox.GetHeight());
    stockpanel.SetLocation( mousepos, ChartObj.DEV_POS);
}
// Draws the tooltip text panel to the chart graphics context
stockpanel.Draw(GetToolTipGraphics());
```

```
        }
    }
}
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 DataToolTip
  Private stockpanel As ChartText
  Private xValues() As ChartCalendar
  Private stockPriceData(,) As Double
  Private stockVolumeData() As Double
  Private NASDAQData() As Double
  Public Sub New(ByVal component As ChartView, ByVal xs() As ChartCalendar, _
           ByVal stockprices(,) As Double, ByVal nasdaqs() As Double, _
           ByVal stockvolumes() As Double)
     MyBase.New(component)
    xValues = xs
    stockPriceData = stockprices
    stockVolumeData = stockvolumes
    NASDAQData = nasdaqs
    stockpanel = GetTextTemplate()
  End Sub 'New


  Public Overrides Sub OnMouseUp(ByVal mouseevent As MouseEventArgs)
     MyBase.OnMouseUp(mouseevent)
 ' Redraws the chart. Since the stockpanel object has not been added to the chart
 ' (using addChartObject) it will not be redrawn when the chart is redrawn
     GetChartObjComponent().UpdateDraw()
  End Sub 'OnMouseUp

  Public Overrides Sub OnMouseDown(ByVal mouseevent As MouseEventArgs)
    Dim mousepos As New Point2D()
```

```
mousepos.SetLocation(mouseevent.X, mouseevent.Y)
MyBase.OnMouseDown(mouseevent)

Dim selectedPlot As ChartPlot = CType(GetSelectedPlotObj(), ChartPlot)
  If Not (selectedPlot Is Nothing) Then
    Dim selectedindex As Integer = GetNearestPoint().NearestPointIndex
    Dim transform As PhysicalCoordinates = GetSelectedCoordinateSystem()
    stockpanel.SetChartObjScale(transform)
    stockpanel.SetLocation(mousepos, ChartObj.DEV_POS)
    stockpanel.SetTextString("Stock Data")
  ' 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 = stockPriceData(0, selectedindex)
     Dim high As Double = stockPriceData(1, selectedindex)
     Dim low As Double = stockPriceData(2, selectedindex)
     Dim close As Double = stockPriceData(3, selectedindex)
     Dim nasdaq As Double = NASDAQData(selectedindex)
     Dim volume As Double = 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, xValues(selectedindex), _
           ChartObj.TIMEDATEFORMAT_STANDARD)
     stockpanel.AddNewLineTextString(timelabel.GetTextString())
     stockpanel.AddNewLineTextString(("Open " + openObj))
     stockpanel.AddNewLineTextString(("High " + highObj))
     stockpanel.AddNewLineTextString(("Low " + lowObj))
     stockpanel.AddNewLineTextString(("Close " + closeObj))
     stockpanel.AddNewLineTextString(("Volume " + volumeObj))
     stockpanel.AddNewLineTextString(("NASDAQ " + nasdaqObj))
     stockpanel.SetChartObjEnable(ChartObj.OBJECT_ENABLE)
     Dim g2 As Graphics = GetToolTipGraphics()
```

```
            ' Precalculates the text bounding box so that the size is
            ' known before it is drawn
            stockpanel.PreCalcTextBoundingBox(g2)
            Dim boundingbox As Rectangle2D = stockpanel.GetTextBox()
            ' Reposition tooltip text box if top of box near top of graph window
            ' You can do the same thing for all four sides of the graph window
            If mousepos.GetY() - boundingbox.GetHeight() < 1 Then
               mousepos.SetLocation(mousepos.GetX(), mousepos.GetY() + _
                  boundingbox.GetHeight())
               stockpanel.SetLocation(mousepos, ChartObj.DEV_POS)
            End If
            ' Draws the tooltip text panel to the chart graphics context
            stockpanel.Draw(GetToolTipGraphics())
         End If
      End Sub 'OnMouseDown
   End Class 'CustomToolTip
```

17. Pie and Ring Charts

PieChart
RingChart

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 datavalues for each pie wedge. The y-values of the dataset explode a pie wedge from its normal centered position.

A ring chart is a variant of the pie chart. Instead an entire circle (or pie) being the basis of the chart, a ring is used instead.

Using the Pie Chart Class

Class PieChart

GraphObj
|
+--ChartPlot
|
+--SimplePlot
|
+-- PieChart
|
+-- RingChart

The **PieChart** and **RingChart** classes extends the **ChartPlot** class and displays pie/ring charts. The x-values of the simple dataset used for data storage specify the pie/ring wedge values. The y-values of the dataset specify the "explode" percentage for each pie/ring wedge.

.

PieChartConstructor

```
[Visual Basic]
Overloads Public Sub New( _
```

```
   ByVal transform As PhysicalCoordinates, _
   ByVal dataset As SimpleDataset, _
   ByVal spiestrings1s As String(), _
   ByVal attribs As ChartAttribute(), _
   ByVal labelinout1 As Integer, _
   ByVal pielabelformat As Integer _
)
[C#]
public PieChart(
   PhysicalCoordinates transform,
   SimpleDataset dataset,
   string[] spiestrings1s,
   ChartAttribute[] attribs,
   int labelinout1,
   int pielabelformat
);
```

RingChart Constructor

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal transform As PhysicalCoordinates, _
   ByVal dataset As SimpleDataset, _
   ByVal spiestrings1s As String(), _
   ByVal attribs As ChartAttribute(), _
   ByVal labelinout1 As Integer, _
   ByVal pielabelformat As Integer _
)
[C#]
public RingChart(
   PhysicalCoordinates transform,
   SimpleDataset dataset,
   string[] spiestrings1s,
   ChartAttribute[] attribs,
   int labelinout1,
   int pielabelformat
);
```

transform	The pie/ring chart is placed in the coordinate system defined by transform.
dataset	The pie/ring chart represents the values in this dataset. The x-values of the simple dataset used for data storage specify the pie/ring wedge values. The y-values of the dataset specify the "explode" percentage for each pie/ring wedge.
spiestrings	An array of strings, size dataset.GetNumberDatapoints(), used as labels for the pie/ring slices.
attribs	An array of **ChartAttribute** objects, size dataset.GetNumberDatapoints() that specify the attributes (outline color and fill color) for each wedge of a pie/ring chart.
labelinout	An array of integer, size dataset.GetNumberDatapoints(), specifying if a specific pie/ring slice text label is drawn inside the

pie/ring slice, or outside of the pie/ring slice. Use one of the constants: PIELABEL_OUTSLICE or PIELABEL_INSLICE.

pielabelformat — All pie/ring slice labels share the same format. Use one of the pie/ring slice label format constants:

PIELABEL_NONE Do not display and pie/ring slice text

PIELABEL_STRING Display only the pie/ring text strings, no numeric values

PIELABEL_NUMVALUE Display the pie/ring numeric value only, no pie text strings.

PIELABEL_STRINGNUMVAL Display the pie/ring text string and numeric value.

A pie/ring chart uses a default **CartesianCoordinates** object. Center it in the window using the **CartesianCoordinates.SetGraphBorderDiagonal** method. Format the text used to label the pie/ring 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/ring wedge from the default value of 0.0 (3:00 position) using the **PieChart.SetStartPieSliceAngle** method. The **PieChart.CalcNearestPoint** method can find the pie/ring wedge nearest a specified point, usually the result of a mouse click. See the **LabeledPieChart** example program.

Simple pie chart (extracted from the example program BigChartDemo, class LabeledPieChart)

[C#]

```
int  numPoints = 5;
String []sPieStrings = {"Technology", "Retail", "Banking",
       "Automotive", "Aerospace"};
ChartAttribute []attribs = new ChartAttribute[5];
Font theFont;
Color []colorArray = {Color.Red, Color.Blue, Color.Cyan,
       Color.Yellow, Color.Green, Color.DarkGray, Color.LightGray,
       Color.Magenta, Color.Orange, Color.Pink};
ChartText techLabel;
ChartText retailLabel;
```

```
ChartText bankLabel;
ChartText aeroLabel;
ChartText autoLabel;

theFont = new Font("Microsoft Sans Serif", 10,  FontStyle.Bold);

double []x1 = new double[numPoints];
double []y1 = new double[numPoints];
int i;

for (i=0; i < numPoints; i++)
{
    attribs[i] = new ChartAttribute (Color.Black, 1,0);
    attribs[i].SetFillColor(colorArray[i]);
}
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;
SimpleDataset Dataset1 = new SimpleDataset("First",x1,y1);
CartesianCoordinates pTransform1 = new CartesianCoordinates();
pTransform1.SetGraphBorderDiagonal(0.1, .2, .9, 0.9) ;

Background background1 = new Background( pTransform1,
        ChartObj.GRAPH_BACKGROUND,Color.FromArgb(0,120,70),
        Color.FromArgb(0,40,30), ChartObj.Y_AXIS);
chartVu.AddChartObject(background1);

PieChart thePlot1 =
        new PieChart(pTransform1, Dataset1, sPieStrings,attribs,
ChartObj.PIELABEL_OUTSLICE, ChartObj.PIELABEL_STRINGNUMVAL);
thePlot1.SetStartPieSliceAngle(-45);

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 numPoints As Integer = 5

Dim sPieStrings As [String]() = {"Technology", "Retail", "Banking", _
       "Automotive", "Aerospace"}
Dim attribs(5) As ChartAttribute
Dim theFont As Font
Dim colorArray As Color() = {Color.Red, Color.Blue, Color.Cyan, _
    Color.Yellow, Color.Green, Color.DarkGray, Color.LightGray, _
    Color.Magenta, Color.Orange, Color.Pink}

Dim techLabel As ChartText
Dim retailLabel As ChartText
Dim bankLabel As ChartText
Dim aeroLabel As ChartText
Dim autoLabel As ChartText

theFont = New Font("Microsoft Sans Serif", 10, FontStyle.Bold)

Dim x1(numPoints-1) As Double
Dim y1(numPoints - 1) As Double
Dim i As Integer

For i = 0 To numPoints - 1
  attribs(i) = New ChartAttribute(Color.Black, 1, 0)
  attribs(i).SetFillColor(colorArray(i))
Next i

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
Dim Dataset1 As New SimpleDataset("First", x1, y1)
Dim pTransform1 As New CartesianCoordinates()
pTransform1.SetGraphBorderDiagonal(0.1, 0.2, 0.9, 0.9)

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)


Dim thePlot1 As New PieChart(pTransform1, Dataset1, sPieStrings, attribs, _
       ChartObj.PIELABEL_OUTSLICE, ChartObj.PIELABEL_STRINGNUMVAL)
thePlot1.SetStartPieSliceAngle(-45)

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)
```

18. Polar and Antenna Plots

PolarPlot
- **PolarLinePlot**
- **PolarScatterPlot**

AntennaPlot
- **AntennaLinePlot**
- **AntennaScatterPlot**
- **AntennaLineMarkerPlot**

Polar charts play an important in engineering applications involving electronics and advanced control systems. Polar charts give a visual interpretation to mathematical problems involving trigonometric functions and complex numbers. Antenna charts are used to display the operating characteristics of antennas.

The **PolarPlot** class is an abstract class representing plot types that use data organized as arrays of x- and y-values, where an x-value represents the magnitude of a point in polar coordinates, and the y-value represents the angle of a point in polar coordinates. Polar plots types include: line plots and scatter plots.

The polar angle values stored as the y-values in the **SimpleDataset** should be in radians. If the raw data is in degrees, convert the data to radians using the ChartSupport.**ToRadians** method.

The **AntennaPlot** class is an abstract class representing plot types that use data organized as arrays of x- and y-values, where an x-value represents the radial value of a point in antenna coordinates, and the y-value represents the angle of a point in antenna coordinates. Antenna plots types include: line plots, scatter plots, line marker plots, and annotations.

The antenna angle values stored as the y-values in the **SimpleDataset** should be specified in degrees. If the raw data is in radians, convert the data to degrees using the ChartSupport.ToDegrees method.

If you plan to create polar charts, you need to also familiarize yourself with the polar charting classes describe in the other chapters of this manual. These are the chapters on coordinate systems (**PolarCoordinates** in Chapter 4), axes (**PolarAxes** in Chapter 7), axis labels (**PolarAxesLabels** in Chapter 8) and grids (**PolarGrids** in Chapter 9). The same is true if you plan to create antenna charts. Refer to the documentation in the chapters on coordinate systems (**AntennaCoordinates** in Chapter 4) , axes (**AntennaAxes** in Chapter 7), axis labels (**AntennaAxesLabels** in Chapter 8) and grids (**AntennaGrids** in Chapter 9).

The **ChartZoom**, **MagniView,** and **MoveCoordinates** classes require a rectangular coordinate system and will not work with polar and antenna charts. The **MoveData** class does work with polar and antenna charts.

Polar Plots

Class PolarLinePlot

GraphObj
|
+--ChartPlot
|
+--PolarPlot
|
+-- PolarLinePlot

The **PolarLinePlot** class is a concrete implementation of the **PolarPlot** class and displays data in a simple line plot format. The lines drawn between adjacent data points use polar coordinate interpolation.

PolarLinePlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal transform As PolarCoordinates, _
   ByVal dataset As SimpleDataset, _
   ByVal attrib As ChartAttribute _
)

[C#]
public PolarLinePlot(
   PolarCoordinates transform,
   SimpleDataset dataset,
   ChartAttribute attrib
);
```

transform	The coordinate system for the new **PolarLinePlot** object.
dataset	The polar line plot represents the polar coordinate values in this dataset. The x-values of the dataset represent the magnitudes of the points and the y-values the polar angles in radians.
attrib	Specifies the attributes (line color) for the line plot.

The polar line plot class interpolates between adjacent data points in polar coordinates and not using straight lines as in the Cartesian coordinate plotting functions. This gives the lines between adjacent data points in a polar plot a curved look.

Polar line plot and scatter plot chart (extracted from the example program BigChartDemo, class PolarLinePlotChart)

[C#]

```
int nump1 = 100;
double []mag1 = new double[nump1];
double []ang1 = new double[nump1];
int i;
for (i=0; i < nump1; i++)
{
    ang1[i] = ChartSupport.ToRadians((double)i * (360.0/ (double)nump1));
    mag1[i] = Math.Abs(30 * (Math.Sin(2*(ang1[i])) * Math.Cos(2 * (ang1[i]))));
}
theFont = new Font("Microsoft Sans Serif", 10,  FontStyle.Bold);
chartVu  = chartView1;

SimpleDataset Dataset1 = new SimpleDataset("First",mag1,ang1);
PolarCoordinates pPolarTransform = new PolarCoordinates();

pPolarTransform.SetGraphBorderDiagonal(0.25, .20, .75, 0.8) ;

Background background =
    new Background( pPolarTransform, ChartObj.GRAPH_BACKGROUND,
    Color.White);
chartVu.AddChartObject(background);

pPolarTransform.AutoScale(Dataset1);
PolarAxes pPolarAxis = pPolarTransform.GetCompatibleAxes();
chartVu.AddChartObject(pPolarAxis);

PolarGrid pPolarGrid = new PolarGrid (pPolarAxis, PolarGrid.GRID_MAJOR);
chartVu.AddChartObject(pPolarGrid);

PolarAxesLabels pPolarAxisLabels = (PolarAxesLabels)
pPolarAxis.GetCompatibleAxesLabels();
chartVu.AddChartObject(pPolarAxisLabels);
```

```
ChartAttribute attrib1 = new ChartAttribute (Color.Blue, 2,0);
PolarLinePlot thePlot1 = new PolarLinePlot(pPolarTransform, Dataset1, attrib1);
chartVu.AddChartObject(thePlot1);

ChartAttribute attrib2 = new ChartAttribute (Color.Red, 1,0,Color.Red);
attrib2.SetFillFlag(true);
PolarScatterPlot thePlot2 =
       new PolarScatterPlot(pPolarTransform,  Dataset1,ChartObj.CIRCLE,attrib2);
chartVu.AddChartObject(thePlot2);
```

[Visual Basic]

```
Dim nump1 As Integer = 100
Dim mag1(nump1 - 1) As Double
Dim ang1(nump1 - 1) As Double

Dim i As Integer
For i = 0 To nump1 - 1
  ang1(i) = ChartSupport.ToRadians((CDbl(i) * (360.0 / CDbl(nump1))))
  mag1(i) = Math.Abs((30 * (Math.Sin((2 * ang1(i))) * Math.Cos((2 * ang1(i))))))
Next i
theFont = New Font("Microsoft Sans Serif", 10, FontStyle.Bold)
chartVu = ChartView1

Dim Dataset1 As New SimpleDataset("First", mag1, ang1)
Dim pPolarTransform As New PolarCoordinates()

pPolarTransform.SetGraphBorderDiagonal(0.25, 0.2, 0.75, 0.8)

Dim background As New Background(pPolarTransform, _
    ChartObj.GRAPH_BACKGROUND, Color.White)
chartVu.AddChartObject(background)

pPolarTransform.AutoScale(Dataset1)
Dim pPolarAxis As PolarAxes = pPolarTransform.GetCompatibleAxes()
chartVu.AddChartObject(pPolarAxis)

Dim pPolarGrid As New PolarGrid(pPolarAxis, PolarGrid.GRID_MAJOR)
chartVu.AddChartObject(pPolarGrid)

Dim pPolarAxisLabels As PolarAxesLabels = _
    CType(pPolarAxis.GetCompatibleAxesLabels(), PolarAxesLabels)
```

```
chartVu.AddChartObject(pPolarAxisLabels)

Dim attrib1 As New ChartAttribute(Color.Blue, 2, 0)
Dim thePlot1 As New PolarLinePlot(pPolarTransform, Dataset1, attrib1)
chartVu.AddChartObject(thePlot1)

 Dim attrib2 As New ChartAttribute(Color.Red, 1, 0, Color.Red)
 attrib2.SetFillFlag(True)
 Dim thePlot2 As New PolarScatterPlot(pPolarTransform, Dataset1, _
       ChartObj.CIRCLE, attrib2)
 chartVu.AddChartObject(thePlot2)
```

Polar Scatter Plots

Class PolarScatterPlot

GraphObj
|
+--ChartPlot
|
+--PolarPlot
|
+-- PolarScatterPlot

The **PolarScatterPlot** class is a concrete implementation of the **PolarPlot** class and displays data in a simple scatter plot format.

PolarScatterPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal transform As PolarCoordinates, _
   ByVal dataset As SimpleDataset, _
   ByVal symtype As Integer, _
   ByVal attrib As ChartAttribute _
)
[C#]
public PolarScatterPlot(
   PolarCoordinates transform,
   SimpleDataset dataset,
   int symtype,
   ChartAttribute attrib
);
```

transform The coordinate system for the new **PolarScatterPlot** object.

dataset	The polar scatter plot represents the polar coordinate values in this dataset. The x-values of the dataset represent the magnitudes of the points and the y-values the polar angles in radians.
symtype	The symbol used in the scatter plot. Use one of the scatter plot symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE.
attrib	Specifies the attributes (size, line and fill color) for the scatter plot.

See previous example for a programming example using PolarScatterPlot.

Antenna Plots

Class AntennaLinePlot

GraphObj
|
+--ChartPlot
|
+--AntennaPlot
|
+-- AntennaLinePlot

The **AntennaLinePlot** class is a concrete implementation of the **AntennaPlot** class and displays data in a simple line plot format. The lines drawn between adjacent data points use antenna coordinate interpolation.

AntennaLinePlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal transform As AntennaCoordinates, _
   ByVal dataset As SimpleDataset, _
   ByVal attrib As ChartAttribute _
)

[C#]
public AntennaLinePlot(
   AntennaCoordinates transform,
   SimpleDataset dataset,
   ChartAttribute attrib
);
```

transform	The coordinate system for the new **AntennaLinePlot** object.
dataset	The antenna line plot represents the antenna coordinate values in this dataset. The x-values of the dataset represent the radial values and the y-values represent the antenna angular values in degrees.
attrib	Specifies the attributes (line color and line style) for the line plot.

The antenna line plot class interpolates between adjacent data points in antenna coordinates and not using straight lines as in the Cartesian coordinate plotting functions. This gives the lines between adjacent data points in a antenna plot a curved look.

Antenna line plot and scatter plot chart (extracted from the example program BigChartDemo.AntennaChart)

[C#]

```
Dataset1 = new SimpleDataset("First", mag1, ang1);
Dataset2 = new SimpleDataset("Second", mag2, ang1);

SimpleDataset[] datasetarray = { Dataset1, Dataset2 };
pAntennaTransform = new AntennaCoordinates();
pAntennaTransform.AutoScale(datasetarray, ChartObj.AUTOAXES_FAR);

pAntennaTransform.SetGraphBorderDiagonal(0.25, .15, .75, 0.85);

Background background = new Background(pAntennaTransform,
ChartObj.GRAPH_BACKGROUND, Color.White);
chartVu.AddChartObject(background);

AntennaAxes pAntennaAxis = pAntennaTransform.GetCompatibleAxes();
pAntennaAxis.LineColor = Color.Black;
chartVu.AddChartObject(pAntennaAxis);

AntennaGrid pAntennaGrid = new AntennaGrid(pAntennaAxis, AntennaGrid.GRID_ALL);
pAntennaGrid.ChartObjAttributes = new ChartAttribute(Color.LightBlue, 1,
DashStyle.Solid);
chartVu.AddChartObject(pAntennaGrid);

AntennaAxesLabels pAntennaAxisLabels =
(AntennaAxesLabels)pAntennaAxis.GetCompatibleAxesLabels();
chartVu.AddChartObject(pAntennaAxisLabels);
```

```
ChartAttribute attrib1 = new ChartAttribute(Color.Red, 1, DashStyle.Solid);
attrib1.SymbolSize = 7;
ChartAttribute attrib2 = new ChartAttribute(Color.Blue, 1, DashStyle.Solid,
Color.Blue);
attrib2.SymbolSize = 7;

ChartAttribute attrib3 = new ChartAttribute(Color.Yellow, 3, DashStyle.Solid,
Color.Yellow);
ChartAttribute attrib4 = new ChartAttribute(Color.MediumPurple, 2, DashStyle.Dash,
Color.MediumPurple);

AntennaLinePlot thePlot1 = new AntennaLinePlot(pAntennaTransform);
thePlot1.InitAntennaLinePlot(Dataset1, attrib1);
chartVu.AddChartObject(thePlot1);

AntennaScatterPlot thePlot2 = new AntennaScatterPlot(pAntennaTransform);
thePlot2.InitAntennaScatterPlot(Dataset1, ChartObj.SQUARE, attrib2);
chartVu.AddChartObject(thePlot2);
```

[Visual Basic]

```
Dataset1 = New SimpleDataset("First", mag1, ang1)
Dataset2 = New SimpleDataset("Second", mag2, ang1)
Dim datasetarray As SimpleDataset() = {Dataset1, Dataset2}
pAntennaTransform = New AntennaCoordinates()
pAntennaTransform.AutoScale(datasetarray, ChartObj.AUTOAXES_FAR)

pAntennaTransform.SetGraphBorderDiagonal(0.25, 0.15, 0.75, 0.85)

Dim background As New Background(pAntennaTransform, ChartObj.GRAPH_BACKGROUND,
Color.White)
chartVu.AddChartObject(background)

Dim pAntennaAxis As AntennaAxes = pAntennaTransform.GetCompatibleAxes()
pAntennaAxis.LineColor = Color.Black
chartVu.AddChartObject(pAntennaAxis)

Dim pAntennaGrid As New AntennaGrid(pAntennaAxis, AntennaGrid.GRID_ALL)
pAntennaGrid.ChartObjAttributes = New ChartAttribute(Color.LightBlue, 1,
DashStyle.Solid)
chartVu.AddChartObject(pAntennaGrid)
```

```
Dim pAntennaAxisLabels As AntennaAxesLabels =
DirectCast(pAntennaAxis.GetCompatibleAxesLabels(), AntennaAxesLabels)
chartVu.AddChartObject(pAntennaAxisLabels)

Dim transparentRed As Color = Color.FromArgb(180, 255, 0, 0)
Dim transparentBlue As Color = Color.FromArgb(180, 0, 0, 255)

Dim attrib1 As New ChartAttribute(transparentRed, 1, DashStyle.Solid)
attrib1.SymbolSize = 7
Dim attrib2 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid, Color.Blue)
attrib2.SymbolSize = 7

Dim thePlot1 As New AntennaLinePlot(pAntennaTransform)
thePlot1.InitAntennaLinePlot(Dataset1, attrib1)
chartVu.AddChartObject(thePlot1)
Dim thePlot2 As New AntennaScatterPlot(pAntennaTransform)
thePlot2.InitAntennaScatterPlot(Dataset1, ChartObj.SQUARE, attrib2)
chartVu.AddChartObject(thePlot2)
```

Class AntennaScatterPlot

GraphObj
|
+--ChartPlot
|
+--AntennaPlot
|
+-- AntennaScatterPlot

The **AntennaScatterPlot** class is a concrete implementation of the **AntennaPlot** class and displays data in a simple scatter plot format.

AntennaScatterPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal transform As AntennaCoordinates, _
   ByVal dataset As SimpleDataset, _
   ByVal symtype As Integer, _
   ByVal attrib As ChartAttribute _
)
```

```
[C#]
public AntennaScatterPlot(
   AntennaCoordinates transform,
   SimpleDataset dataset,
   int symtype,
   ChartAttribute attrib
);
```

transform	The coordinate system for the new **AntennaScatterPlot** object.
dataset	The antenna scatter plot represents the antenna coordinate values in this dataset. The x-values of the dataset represent the radial values of the points and the y-values the angular values in degrees.
symtype	The symbol used in the scatter plot. Use one of the scatter plot symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE.
attrib	Specifies the attributes (size, line and fill color) for the scatter plot.

See previous example for a programming example using AttennaScatterPlot.

Class AntennaLineMarkerPlot

GraphObj
|
+--ChartPlot
|
+--AntennaPlot
|
+-- AntennaLineMarkerPlot

The **AntennaLineMarkerPlot** class is a concrete implementation of the **AntennaPlot** class and displays data in a simple line marker plot format.

AntennaScatterPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal transform As AntennaCoordinates, _
   ByVal dataset As SimpleDataset, _
   ByVal symtype As Integer, _
   ByVal lineattrib As ChartAttribute, _
```

```
   ByVal symbolattrib As ChartAttribute, _
)
[C#]
public AntennaLineMarkerPlot(
   AntennaCoordinates transform,
   SimpleDataset dataset,
   int symtype,
   ChartAttribute lineattrib,
   ChartAttribute symbolattrib,

);
```

transform	The coordinate system for the new **AntennaLineMarkerPlot** object.
dataset	The line marker plot represents the values in this dataset.
symtype	The symbol used in the line marker plot. Use one of the scatter plot symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, CIRCLE.
lineattrib	Specifies the attributes (line color and line style) for the line part of the line marker plot.
symbolattrib	Specifies the attributes (line and fill color) for the symbol part of the line marker plot.

Antenna line marker plot (extracted from the example program Antenna.AntennaLineMarkerChart)

[C#]

```
ChartAttribute attrib1 = new ChartAttribute(Color.Red, 1, DashStyle.Solid);

attrib1.SymbolSize = 7;

ChartAttribute attrib2 = new ChartAttribute(Color.Blue, 1, DashStyle.Solid,
Color.Blue);

attrib2.SymbolSize = 7;


ChartAttribute attrib3 = new ChartAttribute(Color.Yellow, 3, DashStyle.Solid,
Color.Yellow);

ChartAttribute attrib4 = new ChartAttribute(Color.MediumPurple, 2, DashStyle.Dash,
Color.MediumPurple);

AntennaLineMarkerPlot thePlot1 = new AntennaLineMarkerPlot(pAntennaTransform,
Dataset1, attrib1);

chartVu.AddChartObject(thePlot1);
```

```
AntennaLineMarkerPlot thePlot2 = new AntennaLineMarkerPlot(pAntennaTransform,
Dataset2, attrib2);

chartVu.AddChartObject(thePlot2);


AntennaAnnotation thePlot3 = new AntennaAnnotation(pAntennaTransform,
ChartObj.ANTENNA_ANNOTATION_ANGULAR, 180, attrib3);

chartVu.AddChartObject(thePlot3);


AntennaAnnotation thePlot4 = new AntennaAnnotation(pAntennaTransform,
ChartObj.ANTENNA_ANNOTATION_RADIUS, 12, attrib4);

chartVu.AddChartObject(thePlot4);
```

[VB]

```
Dim attrib1 As New ChartAttribute(transparentRed, 1, DashStyle.Solid)

attrib1.SymbolSize = 7

Dim attrib2 As New ChartAttribute(Color.Blue, 1, DashStyle.Solid, Color.Blue)

attrib2.SymbolSize = 7



Dim thePlot1 As New AntennaLineMarkerPlot(pAntennaTransform, Dataset1, attrib1)

chartVu.AddChartObject(thePlot1)


Dim thePlot2 As New AntennaLineMarkerPlot(pAntennaTransform, Dataset2, attrib2)

chartVu.AddChartObject(thePlot2)



Dim attrib3 As New ChartAttribute(Color.Yellow, 3, DashStyle.Solid, Color.Yellow)

Dim attrib4 As New ChartAttribute(Color.MediumPurple, 2, DashStyle.Dash,
Color.MediumPurple)

Dim thePlot3 As New AntennaAnnotation(pAntennaTransform,
ChartObj.ANTENNA_ANNOTATION_ANGULAR, 180, attrib3)

chartVu.AddChartObject(thePlot3)

Dim thePlot4 As New AntennaAnnotation(pAntennaTransform, ChartObj.
ANTENNA_ANNOTATION_RADIUS, 12, attrib4)

chartVu.AddChartObject(thePlot4)
```

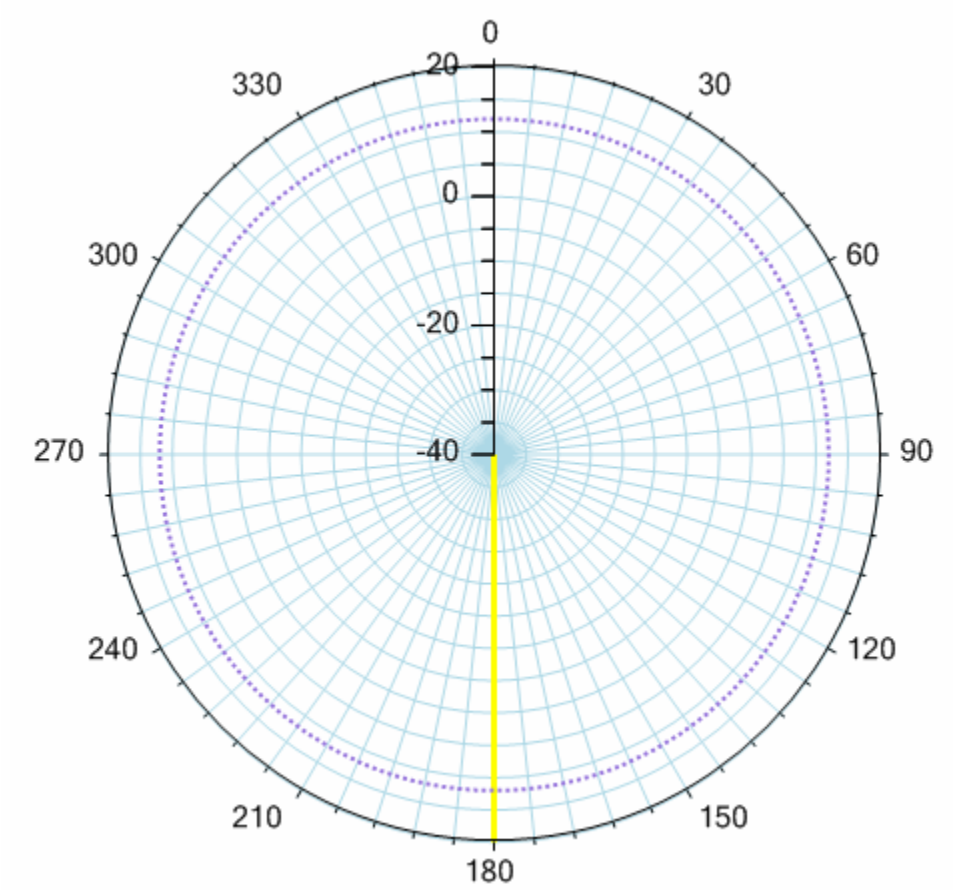
Class AntennaAnnotation

GraphObj

|

+--AntennaAnnotation

The **AntennaAnnotation** class is used to highlight either a specific radius or angular value in an antenna chart. The radius is highlighted using a circle, and the angular value is highlighted using a line draw from the origin to the outer edge of the antenna chart.



Two annotations – a radius annotations (dotted blue line) at the radial value 12, and an angular annotations (solid yellow line) at the angular value 180 degrees.

AntennaAnnotationPlot constructor

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal transform As AntennaCoordinates, _
   ByVal annotationtype As Integer, _
   ByVal value As Double, _
   ByVal attrib As ChartAttribute _
)

[C#]
public AntennaAnnotation(
   AntennaCoordinates transform,
   int annotationtype,
```

```
   double value,
   ChartAttribute attrib
);
```

transform	The coordinate system for the new **AntennaScatterPlot** object.
annotationtype	The annotation type. Use one of the annotation type constants: ANTENNA_ANNOTATION_ANGULAR (draws a radial line at the specified angular value, from the orign to the outer edge of the antenna chart, or ANTENNA_ANNOTATION_RADIUS (draws a circle at the specified radius value).
value	The value of the annotation. For an angular annotation, specify the value in degrees. For a radial annotation, specify a value within the range of the antenna minimum and maximum radial values.
attrib	Specifies the attributes (size, line and fill color) for the annotation.

See previous example for a programming example using AntennaAnnotationPlot.

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, 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.

The **StandardLegend** is a concrete implementation 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 concrete implementation 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 nlayout1mode 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 nlayout1mode As Integer _
)


[C#]
public StandardLegend(
   double rx,
   double ry,
   ChartAttribute attrib,
   int nlayout1mode
);

public StandardLegend(
   double rx,
   double ry,
   double rwidth,
   double rheight,
   ChartAttribute attrib,
   int nlayout1mode
);
```

rx	The x-position, in chart normalized coordinates, of the legend rectangle.
ry	The y-position, in chart normalized coordinates, of the legend rectangle.
rwidth	The width, in chart normalized coordinates, of the legend rectangle.
rheight	The height, in chart normalized coordinates, of the legend rectangle.
attrib	Specifies the outline color and fill color for the legend rectangle.
nlayoutmode	Specifies if the legend has a horizontal, or vertical layout. Use one of the orientation constants: HORIZ_DIR (row major) or VERT_DIR (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
);
```

stext	Specifies the text string for the legend item.
nsymbol	Specifies the symbol for the legend item. Use one of the chart symbol constants: NOSYMBOL, SQUARE, TRIANGLE, DIAMOND, CROSS, PLUS, STAR, LINE, HBAR, VBAR, or CIRCLE.
chartobj	The color and fill attributes for the legend item are copied from the attributes of this **ChartPlot** object.
symbolshape	Specifies a user defined shape to use as the legend item symbol.
attrib	Specifies the **ChartAttribute** object to get the color and fill attributes of the legend item.
thefont	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 BigChartDemo, class LineFill)

[C#]

```
.
.
.
SimpleLinePlot thePlot1 =
   new SimpleLinePlot(pTransform1, Dataset1, attrib1);
chartVu.AddChartObject(thePlot1);
.
.
.
SimpleLinePlot thePlot2 =
    new SimpleLinePlot(pTransform1, Dataset2, attrib2);
chartVu.AddChartObject(thePlot2);
.
.
.
SimpleLinePlot thePlot3 =
        new SimpleLinePlot(pTransform1, Dataset3, profitAttrib);
.
.
.
Font legendFont = new Font("Microsoft Sans Serif", 14, FontStyle.Bold);
ChartAttribute legendAttributes =
    new ChartAttribute(Color.Gray, 1,
    DashStyle.Solid, Color.FromArgb(155,155,155));
legendAttributes.SetFillFlag(true);
legendAttributes.SetLineFlag(true);
StandardLegend legend =
        new StandardLegend(0.2, 0.15, 0.3, 0.3,
            legendAttributes, StandardLegend.VERT_DIR);
legend.AddLegendItem("Expenses",ChartObj.LINE, thePlot1, legendFont);
legend.AddLegendItem("Revenue", ChartObj.LINE, thePlot2, legendFont);
legend.AddLegendItem("Profit", ChartObj.HBAR, thePlot3, legendFont);
legend.AddLegendItem("Loss", ChartObj.HBAR, lossAttrib, legendFont);

chartVu.AddChartObject(legend);
```

[Visual Basic]

```
Dim thePlot1 As SimpleLinePlot = New SimpleLinePlot(pTransform1, _
       Dataset1,  attrib1)
chartVu.AddChartObject(thePlot1)
.
.
.
Dim thePlot2 As SimpleLinePlot = New SimpleLinePlot(pTransform1, _
       Dataset2, attrib2)
chartVu.AddChartObject(thePlot2)
.
.
.
Dim thePlot3 As SimpleLinePlot = New SimpleLinePlot(pTransform1, _
        Dataset3, profitAttrib)
chartVu.AddChartObject(thePlot3)
.
.
.
Dim legendFont As New Font("Microsoft Sans Serif", 14, FontStyle.Bold)
Dim legendAttributes As New ChartAttribute(Color.Gray, 1, _
       DashStyle.Solid, Color.FromArgb(155, 155, 155))
legendAttributes.SetFillFlag(True)
legendAttributes.SetLineFlag(True)
' Undersized legend rectangle tests auto legend rectangle.
Dim legend As New StandardLegend(0.2, 0.15, 0.25, 0.3, _
       legendAttributes, StandardLegend.VERT_DIR)
legend.AddLegendItem("Expenses", ChartObj.LINE, thePlot1, legendFont)
legend.AddLegendItem("Revenue", ChartObj.LINE, thePlot2, legendFont)
legend.AddLegendItem("Profit", ChartObj.HBAR, thePlot3, legendFont)
legend.AddLegendItem("Loss", ChartObj.HBAR, lossAttrib, legendFont)
 chartVu.AddChartObject(legend)
```

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
);
```

plot	The bubble plot object the legend is associated with**.**
rx	The x-position, in chart normalized coordinates, of the legend rectangle.
ry	The y-position, in chart normalized coordinates, of the legend rectangle.
rwidth	The width, in chart normalized coordinates, of the legend rectangle.

rheight The height, in chart normalized coordinates, of the legend rectangle.

attrib Specifies the outline color 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.

rsize 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 BigChartDemo, class BubblePlotChart)

[C#]

```
.
.
.


ChartAttribute attrib1 = new ChartAttribute (Color.Black, 0,DashStyle.Solid);
attrib1.SetFillColor (Color.FromArgb(177, 33, 33));
attrib1.SetFillFlag (true);
BubblePlot thePlot1 = new BubblePlot(pTransform1, Dataset1,
        ChartObj.SIZE_BUBBLE_RADIUS, attrib1);
chartVu.AddChartObject(thePlot1);

Font theTitleFont = new Font("Microsoft Sans Serif", 14, FontStyle.Bold);
ChartTitle mainTitle = new ChartTitle(pTransform1, theTitleFont,
        "DOT COM Bankruptcies and CEO Compensation");
mainTitle.SetTitleType(ChartObj.CHART_HEADER);
mainTitle.SetTitlePosition( ChartObj.CENTER_GRAPH);
chartVu.AddChartObject(mainTitle);

Font theFooterFont = new Font("Microsoft Sans Serif", 10,  FontStyle.Bold);
ChartTitle footer = new ChartTitle(pTransform1, theFooterFont,
"The size (radius or area) of the bubble adds an additional dimension to the
graph.");

ChartAttribute attrib2 =
        new ChartAttribute (Color.FromArgb(177, 33, 33), 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);

BubblePlotLegend legend =
        new BubblePlotLegend(thePlot1, 0.82, 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]

```
.
.
.
Dim attrib1 As New ChartAttribute(Color.Black, 0, DashStyle.Solid)
attrib1.SetFillColor(Color.FromArgb(177, 33, 33))
attrib1.SetFillFlag(True)
Dim thePlot1 As New BubblePlot(pTransform1, Dataset1, _
       ChartObj.SIZE_BUBBLE_RADIUS, attrib1)
chartVu.AddChartObject(thePlot1)

Dim theTitleFont As New Font("Microsoft Sans Serif", 14, FontStyle.Bold)
Dim mainTitle As New ChartTitle(pTransform1, theTitleFont, _
       "DOT COM Bankruptcies and CEO Compensation")
mainTitle.SetTitleType(ChartObj.CHART_HEADER)
mainTitle.SetTitlePosition(ChartObj.CENTER_GRAPH)
chartVu.AddChartObject(mainTitle)

Dim theFooterFont As New Font("Microsoft Sans Serif", 10, FontStyle.Bold)
Dim footer As New ChartTitle(pTransform1, theFooterFont, _
       "The size (radius or area) of the bubble adds an additional dimension to
the graph.")
footer.SetTitleType(ChartObj.CHART_FOOTER)
footer.SetTitlePosition(ChartObj.CENTER_GRAPH)
footer.SetTitleOffset(8)
footer.SetColor(Color.White)
chartVu.AddChartObject(footer)

Dim attrib2 As New ChartAttribute(Color.FromArgb(177, 33, 33), 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)

Dim legend As New BubblePlotLegend(thePlot1, 0.82, 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**
  - **ElapsedTimeLabel**

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.

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 _
)


[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
);
```

transform	Places the text in the coordinate system defined by transform.
tfont	A reference to a **Font** object.
tstring	A reference to a string object.
x	Specifies the x-value of the text position
y	Specifies the y-value of the text position
npostype	Specifies the if the position of the text is specified in physical coordinates, normalized coordinates or .Net CF device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.
xjust	Specifies the horizontal justification of the text. Use one of the text justification constants: JUSTIFY_MIN, JUSTIFY_CENTER or JUSTIFY_MAX.
yjust	Specifies the vertical justification of the text. Use one of the text justification constants: JUSTIFY_MIN, JUSTIFY_CENTER or JUSTIFY_MAX.
rotation	The rotation of the text (Not supported).

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.

ChartText example (extracted from the example program BigChartDemo, class Multiline)

[C#]

```
.
.
.
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 , ChartObj.PHYS_POS);
chartVu.AddChartObject(currentLabel1);

ChartText currentLabel2 = new ChartText(pTransform1, theLabelFont,
               "I(b) = 100uA", 15.5, y1[1,85]+1 , ChartObj.PHYS_POS);
chartVu.AddChartObject(currentLabel2);

ChartText currentLabel3 = new ChartText(pTransform1, theLabelFont,
           "I(b) = 150uA",15.5, y1[2,85]+1 , ChartObj.PHYS_POS);
chartVu.AddChartObject(currentLabel3);

ChartText currentLabel4 = new ChartText(pTransform1, theLabelFont,
           "I(b) = 200uA", 15.5, y1[3,85]+1 , ChartObj.PHYS_POS);
chartVu.AddChartObject(currentLabel4);

ChartText currentLabel5 = new ChartText(pTransform1, theLabelFont,
           "I(b) = 250uA", 15.5, y1[4,85]+1 , ChartObj.PHYS_POS);
chartVu.AddChartObject(currentLabel5);
```

[Visual Basic]

```
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, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel1)

Dim currentLabel2 As New ChartText(pTransform1, theLabelFont, "I(b) = 100uA", _
       15.5, y1(1, 85) + 1, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel2)

Dim currentLabel3 As New ChartText(pTransform1, theLabelFont, "I(b) = 150uA", _
        15.5, y1(2, 85) + 1, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel3)
```

```
Dim currentLabel4 As New ChartText(pTransform1, theLabelFont, "I(b) = 200uA", _
        15.5, y1(3, 85) + 1, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel4)

Dim currentLabel5 As New ChartText(pTransform1, theLabelFont, "I(b) = 250uA", _
       15.5, y1(4, 85) + 1, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel5)

Dim currentLabel6 As New ChartText(pTransform1, theLabelFont, "I(b) = 300uA", _
       15.5, y1(5, 85) + 1, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel6)

Dim currentLabel7 As New ChartText(pTransform1, theLabelFont, "I(b) = 350uA", _
       15.5, y1(6, 85) + 1, ChartObj.PHYS_POS)
chartVu.AddChartObject(currentLabel7)

Dim regionLabel As New ChartText(pTransform1, theLabelFont, _
       "Linear" + ControlChars.Lf + "Region", 4.0, 40, ChartObj.PHYS_POS)
chartVu.AddChartObject(regionLabel)
```

ChartText time coordinates example (extracted from the example program BigChartDemo, class LineGapChart)

[C#]

```
Font theLabelFont = new Font("Microsoft Sans Serif", 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("Microsoft Sans Serif", 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 ntitletype As Integer, _
   ByVal ntitlepos As Integer _
)


[C#]
public ChartTitle(
   PhysicalCoordinates transform,
   Font tfont,
   string tstring
);

public ChartTitle(
   PhysicalCoordinates transform,
   Font tfont,
   string tstring,
   int ntitletype,
   int ntitlepos
);
```

transform	Places the text in the coordinate system defined by transform.
tfont	A reference to a **Font** object.
tstring	A reference to a string object.
ntitletype	The title can be a header, subhead or footer. Use one of the title type constants: CHART_HEADER, CHART_SUBHEAD or CHART_FOOTER.

ntitlepos The title can be centered with respect to the entire graph area, or the plot area. Use one of the title position constants: CENTER_GRAPH or CENTER_PLOT.

ChartTitle example (extracted from the example program BigChartDemo, class LineFill)

[C#]

```
.
.
.
Font theTitleFont = new Font("Microsoft Sans Serif", 16, FontStyle.Bold);
mainTitle = new ChartTitle(pTransform1, theTitleFont, "Profits are Expected to
Rise");
mainTitle.SetTitleType(ChartObj.CHART_HEADER);
mainTitle.SetTitlePosition( ChartObj.CENTER_GRAPH);
mainTitle.SetColor(Color.White);
chartVu.AddChartObject(mainTitle);
Font theFooterFont = new Font("Microsoft Sans Serif", 9, FontStyle.Bold);
footer = new ChartTitle(pTransform1, theFooterFont,
    "Graphs can have background gradients, 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);
```

[Visual Basic]

```
Dim theTitleFont As New Font("Microsoft Sans Serif", 16, FontStyle.Bold)
mainTitle = New ChartTitle(pTransform1, 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("Microsoft Sans Serif", 10, FontStyle.Bold)
footer = New ChartTitle(pTransform1, theFooterFont, _
       "Graphs can have background gradients 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 both the x-axis titles and y-axis titles (.Net CF does not support rotated text). 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.

ChartTitle example (extracted from the example program BigChartDemo, class 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("Microsoft Sans Serif", 12, FontStyle.Bold);
AxisTitle yaxistitle = new AxisTitle( yAxis, titleFont, "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("Microsoft Sans Serif", 12, FontStyle.Bold)
Dim yaxistitle As New AxisTitle(yAxis, titleFont, "Score")
chartVu.AddChartObject(yaxistitle)

Dim xaxistitle As New AxisTitle(xAxis, titleFont, "Student #")
chartVu.AddChartObject(xaxistitle)
```

Numeric, Time, Elapsed Time and String Label Classes

Class ChartLabel

ChartText

|

+-- ChartLabel

|

+--StringLabel

|

+--TimeLabel

|

+--ElapsedTimeLabel

|

+--NumericLabel

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. Justify the text vertically and horizontally.

ChartLabel contructors

```
[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 _
)


[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
);


NumericLabel(PhysicalCoordinates transform,
                     Font tfont, double initialvalue,
                     double x, double y, int npostype,
                     int nnumformat, int ndecimal)
```

transform	Places the text in the coordinate system defined by transform.
tfont	A reference to a Font object.
initialvalue	The initial value of the numeric label.
x	Specifies the x-value of the text position
y	Specifies the y-value of the text position
npostype	Specifies the if the position of the text is specified in physical coordinates, normalized coordinates or .Net CF device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.
nnumformat	Specifies the numeric format of the label. Use one of the numeric format constants : DECIMALFORMAT, SCIENTIFICFORMAT, BUSINESSFORMAT, ENGINEERINGFORMAT, PERCENTFORMAT, CURRENCEYFORMAT and EXPONENTFORMAT.
ndecimal	The number of digits to display to the right of the decimal point.
xjust	Specifies the horizontal justification of the text. Use one of the text justification constants:

	JUSTIFY_MIN, JUSTIFY_CENTER or JUSTIFY_MAX.
yjust	Specifies the vertical justification of the text. Use one of the text justification constants: JUSTIFY_MIN, JUSTIFY_CENTER or JUSTIFY_MAX.
rotation	The rotation of the text (Not supported).

The **TimeLabel, ElapsedTimeLabel** and **StringLabel** classes are similar, unique properties for each are listed below.

TimeLabel contructors

```
[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 _
)
```

```
[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
);
```

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.

ElapsedTimeLabel constructors

```
[Visual Basic]
Overloads Public Sub New( _
   ByVal transform As PhysicalCoordinates, _
   ByVal timespan As TimeSpan, _
   ByVal timeformat As Integer _
)
```

```
Visual Basic (Declaration)
Public Sub New ( _
    transform As PhysicalCoordinates, _
    tfont As Font, _
    timespan As TimeSpan, _
    x As Double, _
    y As Double, _
    npostype As Integer, _
    timeformat As Integer, _
    xjust As Integer, _
    yjust As Integer, _
    rotation As Double _
)
```

```
C#
public ElapsedTimeLabel (
   PhysicalCoordinates transform,
   ChartCalendar date,
   int timeformat
);
```

```
public ElapsedTimeLabel(
    PhysicalCoordinates transform,
    Font tfont,
    TimeSpan timespan,
    double x,
    double y,
    int npostype,
```

```
    int timeformat,
    int xjust,
    int yjust,
    double rotation
)
```

timespan The time span value used to initialize the label.

timeformat The format used to convert the time span value to a text stringUse one of the TIMEDATAFORMAT_ constants: TIMEDATEFORMAT_NONE, TIMEDATEFORMAT_24HMS, TIMEDATEFORMAT_24HM, TIMEDATEFORMAT_MS.

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 _
)


[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
);
```

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.

NumericLabel and StringLabel example (extracted from the example program BigChartDemo, class MoveDatapoint)

[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);
```

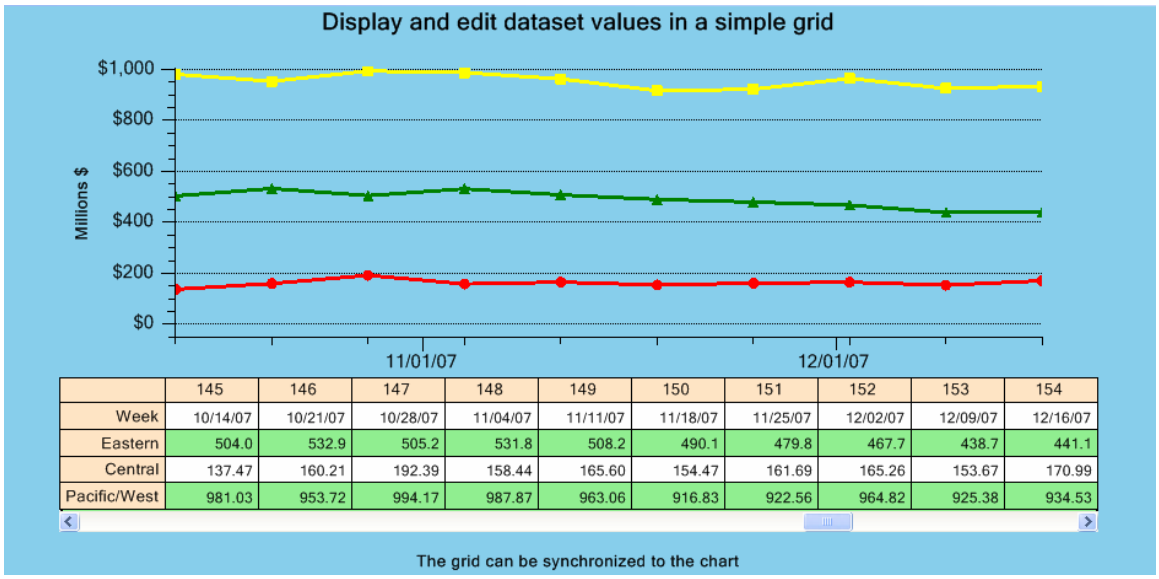
[Visual Basic]

```
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)
hartVu.AddChartObject(class2Label)

class2AverageLabel = New NumericLabel(pTransform1, subheadFont, _
        class2Average, 0.9, 0.55, ChartObj.NORM_GRAPH_POS, _
        ChartObj.DECIMALFORMAT, 1)
chartVu.AddChartObject(class2AverageLabel)
```

21. Dataset Viewers

DatasetViewer

Charts and data grids are probably the two most popular ways to display numeric data. While .Net includes some basic grid controls, they tend to be hard to use and database oriented. We have created our own grid class that integrates with our chart dataset classes. The **DatasetViewer** can display simple numeric and time-based datasets (**SimpleDataset**, **TimeSimpleDataset, ElapsedTimeSimpleDataset**) and group numeric and time-based datasets (**GroupDataset**, **TimeGroupsDataset, ElapsedTimeGroupDataset**). When a **DatasetViewer** is added to a chart, it can be printed as part of that chart. Background colors, row and column headers, can be customized. The **DatasetViewer** can be scrolled, updated in real-time, and synchronized to the chart, so that scrolling of the DatasetViewer can scroll the chart.



	145	146	147	148	149	150	151	152	153	154
Week	10/14/07	10/21/07	10/28/07	11/04/07	11/11/07	11/18/07	11/25/07	12/02/07	12/09/07	12/16/07
Eastern	504.0	532.9	505.2	531.8	508.2	490.1	479.8	467.7	438.7	441.1
Central	137.47	160.21	192.39	158.44	165.60	154.47	161.69	165.26	153.67	170.99
Pacific/West	981.03	953.72	994.17	987.87	963.06	916.83	922.56	964.82	925.38	934.53

A DatasetViewer displaying three TimeSimpleDatasets

Class DatasetViewer

ChartView

|

+--DatasetViewer

The **DatasetViewer** is a **ChartView** derived object and as such is an independent **UserControl** object. Use it to view one or more datasets in a chart. Since it is usually not possible or practical to display the entire dataset, the **DatasetViewer** windows a rectangular section of the dataset for display. Scroll bars are used to scroll the rows and columns of the dataset. The **DatasetViewer** constructor defines the size, position, source matrix, the number of rows and columns of the **DatasetViewer** grid, and the starting position of the **DatasetViewer** scrollbar.

DatasetViewer constructor

Visual Basic (Declaration)

```
Public Sub New ( _
    chartvu As ChartView, _
    transform As PhysicalCoordinates, _
    posrect As Rectangle2D, _
    dataset As ChartDataset, _
    rows As Integer, _
    cols As Integer, _
    start As Integer _
)
```

C#

```
public DatasetViewer(
    ChartView chartvu,
    PhysicalCoordinates transform,
    Rectangle2D posrect,
    ChartDataset dataset,
    int rows,
    int cols,
    int start
)
```

chartvu	The **ChartView** object the **DatasetViewer** is placed in.
transform	The coordinate system the **DatasetViewer** is placed in.
posrect	A positioning rectangle (using normalized chart coordinates) for the dataset viewer, use null if not used.
dataset	A simple, or group, dataset to add to the dataset viewer.
rows	Number of rows to display
cols	Number of columns to display.
start	Starting column of the dataset viewer.

Set unique fonts for the column headers, row headers and grid cells using the ColumnHeaderFont, RowHearderFont and GridCellFont properties.

Turn on the edit feature of the grid cells using the EnableEdit property. Turn on the striped background color of the grid cells using the UseStripedGridBackground property.

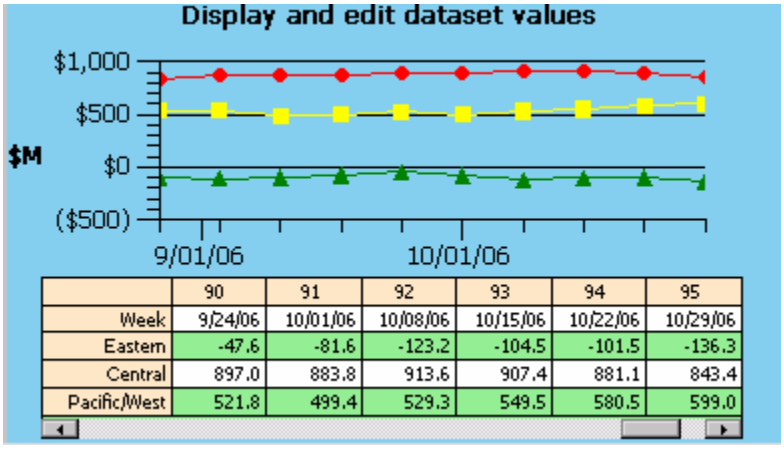
Foreground and background attributes of the column headers, row headers and grid cells can be set using the ColumnHeaderAttribute, RowHeaderAttribute, GridAttribute, and AltGridAttribute properties.

You can add multiple datasets to a **DatasetViewer** using the DatasetViewer.AddDataset method. When adding additional datasets, it only adds the y-values of the dataset. It is assumed the x-values of the datasets are the same; otherwise, the columns would lose synchronization.

The row header string for the first grid row, the x-values, is picked up from the first dataset's XString property. If that is null, "X-Values" is displayed for numeric x-values, and "Time" for time-based x-values. Subsequent row header strings, for the y-values, are picked up from the main title string of each associated dataset. In the case of group datasets with multiple y-values for each x-value, row header strings are picked up from the datasets GroupStrings property, which stores one string for each group in the dataset.

You can change the default orientation of the **DatasetViewer** by calling a version of the **DatasetViewer** constructor that has an orientation property as the last parameter. See the NewDemosRev2.VerticalDatasetViewerChart for an example.

Simple DatasetViewer example (extracted from the example program NewDemosRev2.SimpleDatasetViewer)



	90	91	92	93	94	95
Week	9/24/06	10/01/06	10/08/06	10/15/06	10/22/06	10/29/06
Eastern	-47.6	-81.6	-123.2	-104.5	-101.5	-136.3
Central	897.0	883.8	913.6	907.4	881.1	843.4
Pacific/West	521.8	499.4	529.3	549.5	580.5	599.0

A DatasetViewer displaying three TimeSimpleDatasets

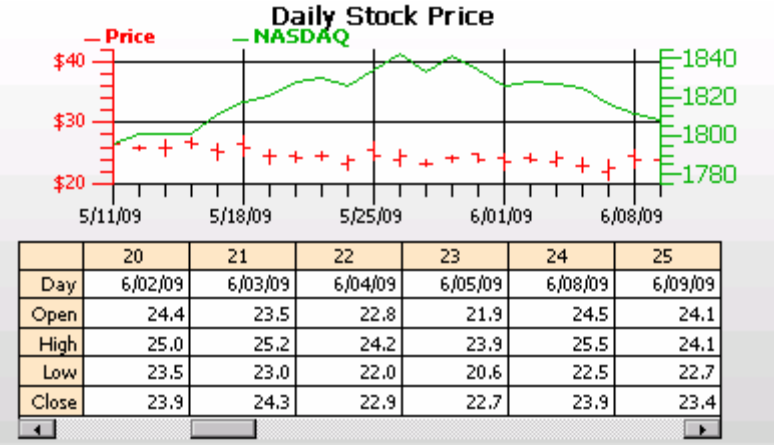
[C#]

```
Rectangle2D posrect = new Rectangle2D(0.05, 0.63, 0.9, 0.36);
DatasetViewer datasetViewer1 =
   new DatasetViewer(chartVu, pTransform1, posrect, Dataset1, 2, 6, 0);
datasetViewer1.EnableEdit(true);
datasetViewer1.DatasetTable.TableBackgroundMode =
   ChartGeneralizedTableDisplay.TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL;
datasetViewer1.UseStripedGridBackground = true;
datasetViewer1.RowHeaderFont =
    new Font("Microsoft Sans Serif", 7, FontStyle.Regular);
datasetViewer1.ColumnHeaderFont =
    new Font("Microsoft Sans Serif", 7, FontStyle.Regular);
datasetViewer1.GridCellFont =
    new Font("Microsoft Sans Serif", 7, FontStyle.Regular);
datasetViewer1.SyncChart = true;
```

[Visual Basic]

```
Dim posrect As New Rectangle2D (0.05, 0.63, 0.9, 0.36)
Dim rows As Integer = 4, columns As Integer = 6, _
    startindex As Integer = initialstartindex
Dim datasetViewer1 As New DatasetViewer(chartVu, pTransform1, posrect, _
    Dataset1, rows, columns,  startindex)
datasetViewer1.AddDataset(Dataset2)
datasetViewer1.AddDataset(Dataset3)
datasetViewer1.EnableEdit(True)
datasetViewer1.DatasetTable.TableBackgroundMode = _
    ChartGeneralizedTableDisplay.TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL
datasetViewer1.UseStripedGridBackground = True
datasetViewer1.RowHeaderFont = _
    New Font("Microsoft Sans Serif", 7, FontStyle.Regular)
datasetViewer1.ColumnHeaderFont = _
    New Font("Microsoft Sans Serif", 7, FontStyle.Regular)
datasetViewer1.GridCellFont = _
    New Font("Microsoft Sans Serif", 7, FontStyle.Regular)
datasetViewer1.SyncChart = True
```

Group DatasetViewer example (extracted from the example program NewDemosRev2.GroupDatasetViewer)



	20	21	22	23	24	25
Day	6/02/09	6/03/09	6/04/09	6/05/09	6/08/09	6/09/09
Open	24.4	23.5	22.8	21.9	24.5	24.1
High	25.0	25.2	24.2	23.9	25.5	24.1
Low	23.5	23.0	22.0	20.6	22.5	22.7
Close	23.9	24.3	22.9	22.7	23.9	23.4

A **DatasetViewer** displaying a **TimeGroupDataset** display open-high-low-close data

[C#]

```
Rectangle2D posrect = new Rectangle2D(0.03, 0.55, 0.9, 0.44);
DatasetViewer.DefaultFont = new Font(FontFamily.GenericSansSerif, 7,
FontStyle.Regular);
DatasetViewer datasetViewer1 = new DatasetViewer(chartVu, pTransform1, posrect,
Dataset1, 5, 6, 0);
datasetViewer1.EnableEdit(true);
datasetViewer1.DatasetTable.TableBackgroundMode = _
  ChartGeneralizedTableDisplay.TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL;
datasetViewer1.SyncChart = true;
datasetViewer1.SetArrayFormat(0, ChartObj.TIMEDATEFORMAT_MDY);
datasetViewer1.TransformList.Add(pTransform2);
datasetViewer1.TransformList.Add(pTransform3);
```

[Visual Basic]

```
Dim posrect As New Rectangle2D(0.03, 0.55, 0.9, 0.44)
DatasetViewer.DefaultFont = _
    New Font(FontFamily.GenericSansSerif, 7, FontStyle.Regular)
Dim datasetViewer1 As New DatasetViewer(chartVu, pTransform1, _
    posrect, Dataset1, 5, 6, 0)
datasetViewer1.EnableEdit(True)
datasetViewer1.DatasetTable.TableBackgroundMode =
ChartGeneralizedTableDisplay.TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL
datasetViewer1.SyncChart = True
datasetViewer1.SetArrayFormat(0, ChartObj.TIMEDATEFORMAT_MDY)
```

```
datasetViewer1.TransformList.Add(pTransform2)
datasetViewer1.TransformList.Add(pTransform3)
```

Vertical Orientation DatasetViewer example (extracted from the example program NewDemosRev2. VerticalDatasetViewerChart)

[C#]

```
Rectangle2D posrect = new Rectangle2D(0.675, 0.1, 0.32, 0.7);
int rows = 10, columns = 2, startindex = 0;
DatasetViewer datasetViewer1 = new DatasetViewer(chartVu, pTransform1, posrect,
Dataset1, rows, columns, startindex, ChartObj.VERT_DIR);
datasetViewer1.EnableEdit(true);
datasetViewer1.DatasetTable.TableBackgroundMode =
ChartGeneralizedTableDisplay.TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL;
datasetViewer1.UseStripedGridBackground = true;
// Need to override default numeric format template with elapsed time template
ElapsedTimeLabel etimetemplate = new ElapsedTimeLabel(new Font("Microsoft Sans
Serif", 7, FontStyle.Regular), ChartObj.TIMEDATEFORMAT_24HMS);
datasetViewer1.SetFormatTemplate(1, etimetemplate);
```

[Visual Basic]

```
Dim posrect As New Rectangle2D(0.675, 0.1, 0.32, 0.7)
Dim rows As Integer = 10, columns As Integer = 2, startindex As Integer = 0
Dim datasetViewer1 As New DatasetViewer(chartVu, pTransform1, posrect, _
    Dataset1, rows, columns, startindex, ChartObj.VERT_DIR)
datasetViewer1.EnableEdit(True)
datasetViewer1.DatasetTable.TableBackgroundMode = _
  ChartGeneralizedTableDisplay.TABLE_SINGLE_COLOR_BACKGROUND_GRIDCELL
datasetViewer1.UseStripedGridBackground = True

// Need to override default numeric format template with elapsed time template
Dim etimetemplate as ElapsedTimeLabel = new ElapsedTimeLabel(new Font("Microsoft
Sans Serif", 7, FontStyle.Regular), ChartObj.TIMEDATEFORMAT_24HMS)
datasetViewer1.SetFormatTemplate(1, etimetemplate)
```

22.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 arcs, rectangles and other classes, a generalized class has been created, **ChartShape**, that can place and display in a chart any object that can be expressed as a **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, or it can be sized to fill the plot area or graph area and used as a background object.

Generic Shape Class

Class ChartShape

GraphObj
|
+--ChartShape

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 _
)
[C#]
public ChartShape(
   PhysicalCoordinates transform,
   GraphicsPath ashape,
   int shapecoordstype,
   double x,
   double y,
   int npositiontype,
   int rotation
);
```

transform	The shape object is placed in the coordinate system defined by transform.
ashape	A reference to a **GraphicsPath** object.
shapecoordstype	Specifies if the coordinate system defining the shape is specified in physical coordinates, normalized coordinates or .Net CF device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.
x	Specifies the x-value of the shape position.
y	Specifies the y-value of the shape position.
npostype	Specifies the if the position of the shape is specified in physical coordinates, normalized coordinates or .Net CF device coordinates. Use one of the position constants: DEV_POS, PHYS_POS, NORM_GRAPH_POS, NORM_PLOT_POS.
rotation	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 BigChartDemo, class MultiLine)

[C#]

```
ChartView chartVu  = chartView1;
.
.
```

```
.
Color alphaColor =  Color.FromArgb(170, 100, 50);
ChartAttribute attrib2 =
    new ChartAttribute (alphaColor, 1,DashStyle.Solid, alphaColor);
attrib2.SetFillFlag(true);
Rectangle2D linearRegionRect = new Rectangle2D(0.1,0.1,1.5,50);
GraphicsPath rectpath = new GraphicsPath();
rectpath.AddRectangle(linearRegionRect.GetRectangleF());
ChartShape linearRegionShape =
    new ChartShape(pTransform1, rectpath, ChartObj.PHYS_POS,
                0.0, 0.0, ChartObj.PHYS_POS,0);
linearRegionShape.SetChartObjAttributes(attrib2);
chartVu.AddChartObject(linearRegionShape);
```

[Visual Basic]

```
Dim alphaColor As Color = Color.FromArgb(170, 100, 50)
Dim attrib2 As New ChartAttribute(alphaColor, 1, DashStyle.Solid, alphaColor)
attrib2.SetFillFlag(True)
Dim linearRegionRect As New Rectangle2D(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, ChartObj.PHYS_POS, 0)
linearRegionShape.SetChartObjAttributes(attrib2)
chartVu.AddChartObject(linearRegionShape)
```

ChartShape example (extracted from the example program BigChartDemo, class LabeledDatapoints)

[C#]

```
GraphicsPath titleLine = new GraphicsPath();
titleLine.AddLine(0.1f,0.1f, 0.9f,0.1f);
ChartShape titleLineShape = new ChartShape(pTransform1, titleLine,
           ChartObj.NORM_GRAPH_POS, 0.0, 0.0, ChartObj.NORM_GRAPH_POS,0);
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(pTransform1, titleLine, _
```

```
        ChartObj.NORM_GRAPH_POS, 0.0, 0.0, ChartObj.NORM_GRAPH_POS, 0)
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 or it can be sized to fill the plot area or graph area and used as a background object.

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 npostype As Integer, _
   ByVal rotation As Integer _
)
[C#]
public ChartImage(
   PhysicalCoordinates transform,
   Image aimage,
   double x,
   double y,
   int npostype,
   int rotation
);
```

transform	The coordinate system for the new **ChartImage** object.
aimage	A reference to the **Image** object that is to be placed in the chart.
x	The x-value for the position of the image in the chart.
y	The y-value for the position of the image in the chart.
npostype	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.
rotation	The rotation of the image specified in degrees (Not supported).

ChartImage example (extracted from the example program BigChartDemo, class CloudsBackgroundPlot)

[C#]

```
private Bitmap GetImageFile(String imagefilename)
{
    Bitmap result = null;
    try
    {
        result = new Bitmap(imagefilename);
    }
    catch (System.ArgumentException )
    {
        result = null;
    }
    return result;
}
.
.
.
Bitmap aImage = GetImageFile( "/Program Files/BigChartDemo/ChartClouds.jpg");
 if (aImage != null)
{
    ChartImage chartImage = new ChartImage( pTransform1,
                aImage, 0, 0,  ChartObj.NORM_GRAPH_POS, 0 );
    chartImage.SetSizeMode(ChartObj.COORD_SIZE);
    chartImage.SetImageSize(new Dimension(1,1));
    chartVu.AddChartObject(chartImage);
}
```

[Visual Basic]

```
Private Function GetImageFile(ByVal imagefilename As String) As Bitmap
 Dim result As Bitmap
    Try
       result = New Bitmap(imagefilename)
    Catch err As System.ArgumentException
```

```
        result = Nothing
    End Try
  Return result
End Function
.
.
.
.
Dim aImage As Bitmap = GetImageFile("/Program Files/BigChartDemo/ChartClouds.jpg")

Dim aImage As New Bitmap(filename)
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 Dimension(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
);
```

arrowshafthalfwidth	Sets the half-width of the arrow shaft. (default 1
arrayshaftlength	Sets the length of the arrow shaft. (default 7)
arrowheadhalfwidth	Sets the half-width of the arrow head. (default 2)
arrowheadlength	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 BigChartDemo, class MultiLine)

[C#]

```
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, ChartObj.PHYS_POS,195);
arrowShape.SetChartObjAttributes(arrowAttrib);
chartVu.AddChartObject(arrowShape);
```

[Visual Basic]

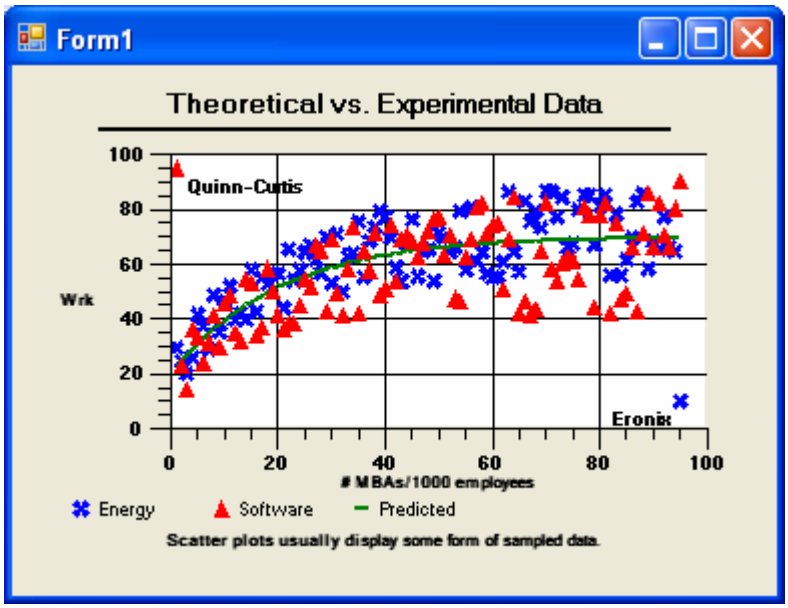
```
Dim regionArrow As New Arrow(1, 40, 6, 15)
Dim arrowAttrib As New ChartAttribute(Color.Black, 1, _
        DashStyle.Solid, Color.Black)
arrowAttrib.SetFillFlag(True)
Dim arrowShape As New ChartShape(pTransform1, regionArrow.GetArrowShape(), _
```

```
        ChartObj.DEV_POS, 1.5, 40.0, ChartObj.PHYS_POS, 195)
arrowShape.SetChartObjAttributes(arrowAttrib)
chartVu.AddChartObject(arrowShape)
```

23 Using QCChart2D CF for the .Net Compact Framework to Create Windows Applications

The primary view class of the **QCChart2D CF** 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 **Control** class. The main example program for the **QCChart2D CF** library is the **BigChartDemo** program. This program incorporates 50 different examples into a single program, selectable as items in a menu. While this is a good program for us to use as a test suite to test and demonstrate the software, it probably does not reflect you own intended use.

The following steps describe techniques you can use to incorporate the **QCChart2D CF** classes into your program. These are 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.



UserChartExample1, UserChartExample2 and UserChartExample3 all produce this chart.

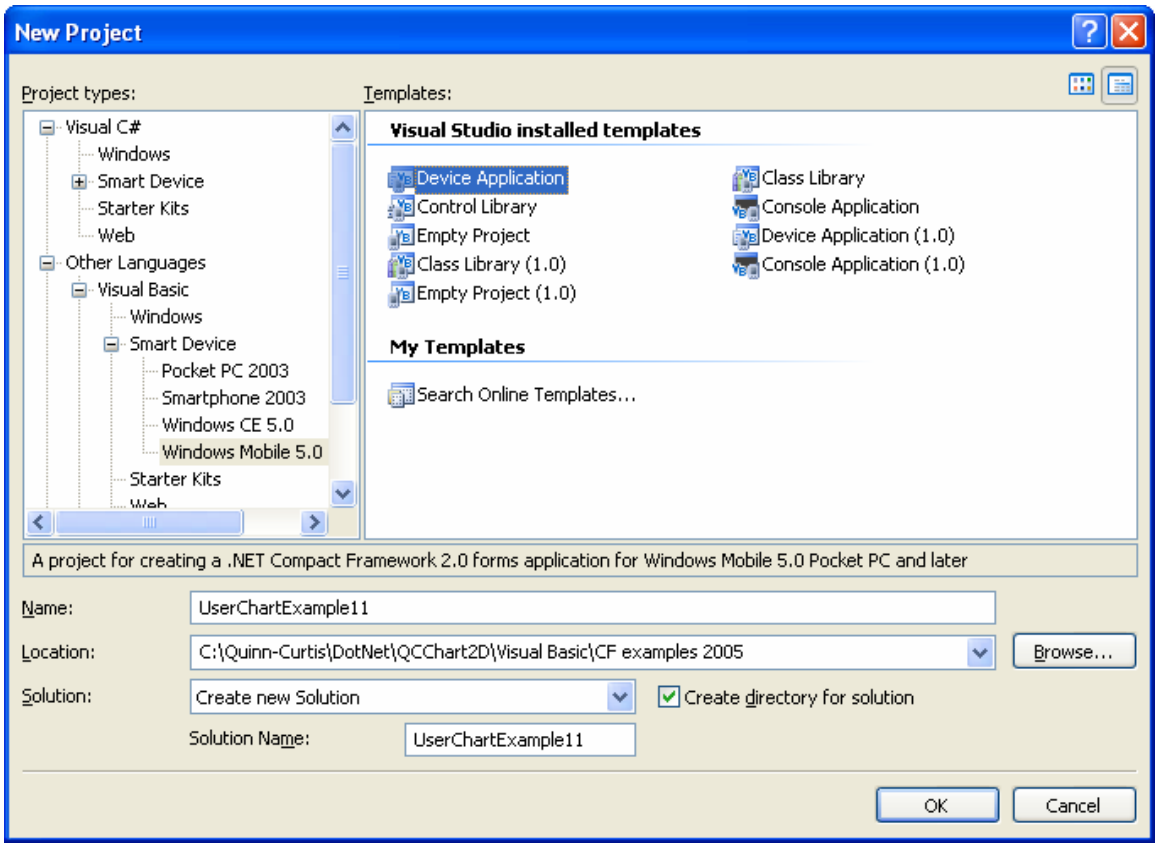
Visual Basic for .Net Compact Framework

First, if this is the first **.Net Compact Framework** program you have every created, make a few practice application programs using the Visual Studio defaults. Don't try to add graphics to an application until you are able to create a simple .**Net Compact Framework** applications using the **New Project (File | New | Project)** application wizard.

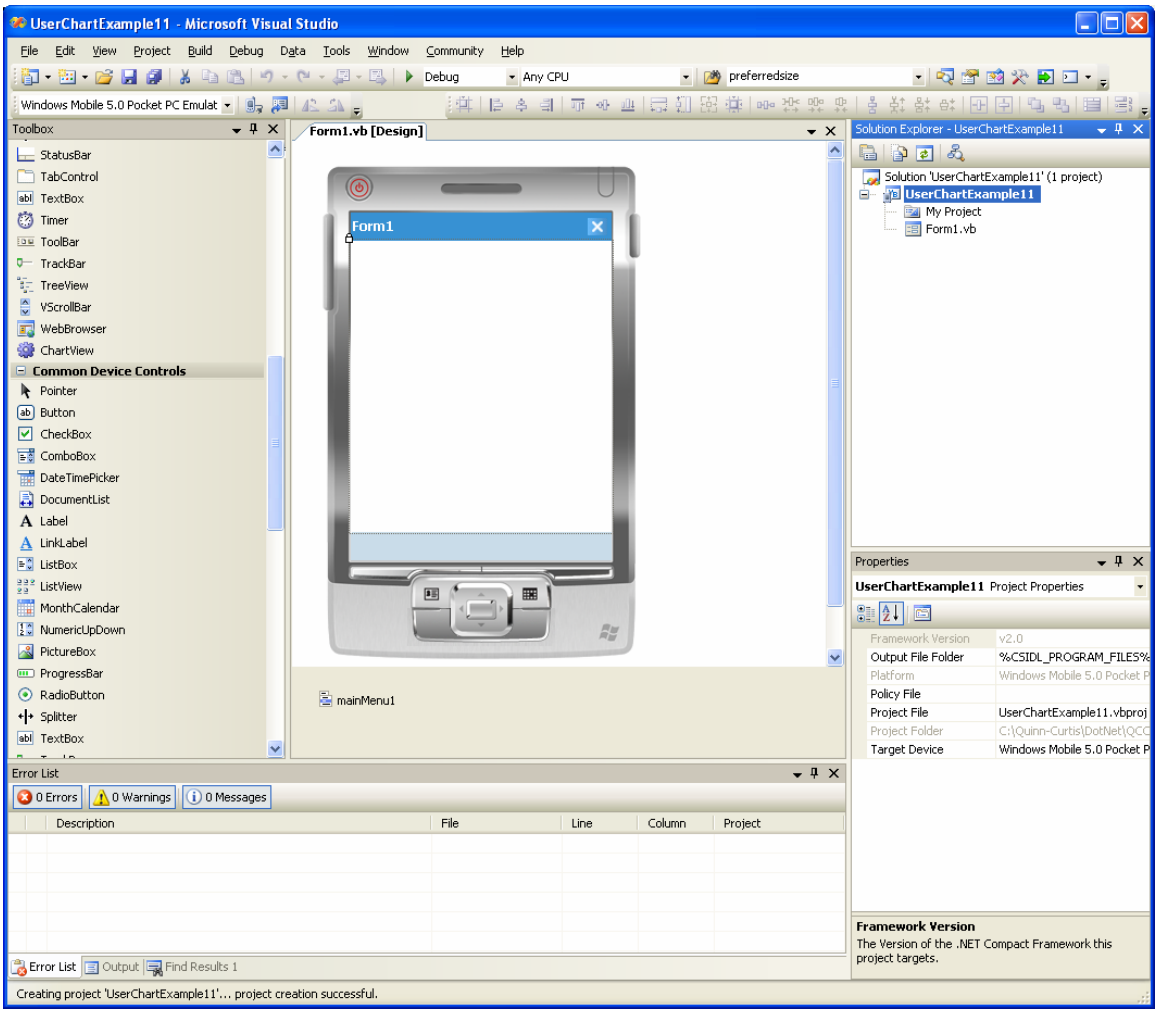
- You start the **New Project** application wizard by selecting **File | New | Project**, bringing up the **New Project** dialog box.

- From this dialog, select **Visual Basic Projects | Smart Device | Windows Mobile 5.0** folder on the left, and the **Device Application** template on the right. The default Device Application targets a .Net CF 2.0 device. Do **NOT** target a .Net 1.0 device by selecting the Device Application (1.0), since this software is not compatible with .Net 1.0 and 1.1.

- Assign a name to the application in the name box, either the default (**DeviceApplication1**), or your own pick (**UserChartExample11** in the example below). Select a location, which for our examples is the folder C:\Quinn-Curtis\DotNet\QCChart2D\Visual Basic\CF examples 2005.



- Click OK and you end up looking at the basic template for a simple application.



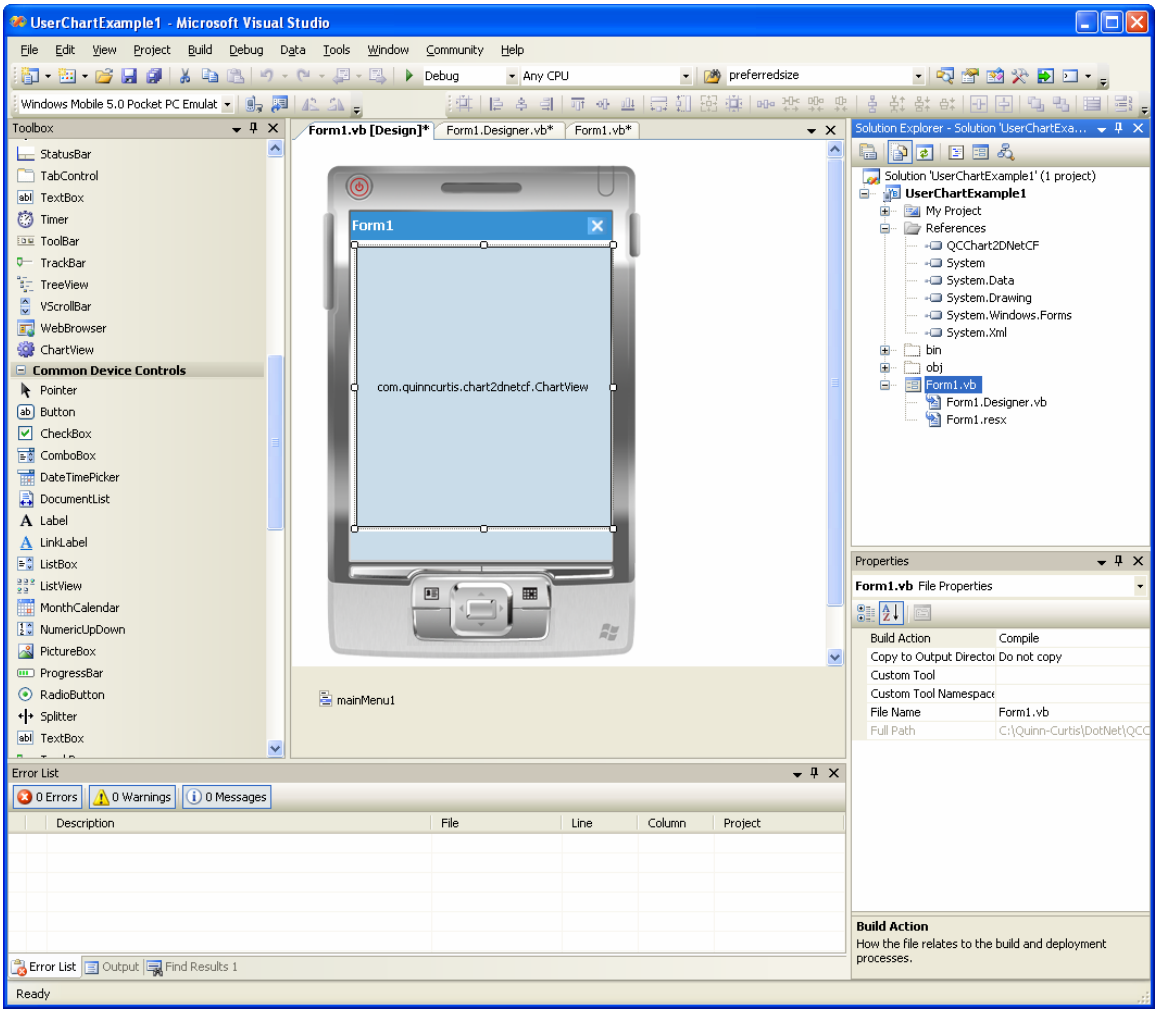
.

The previous steps apply to all **.Net Compact Framework** applications. The following steps are specific to adding **QCChart2D CF** charts to your application. For purposes of this example, the chart will placed in the initial, default form.

Using ChartView with the Visual Studio Design Mode

- **Add the ChartView control to the Toolbox.** If it isn't there already, add the **ChartView** UserControl to the Visual Studio Toolbox. To add the **ChartView** control to the VS Toolbox, right click on the Toolbox and select **Choose Items**. From the Choose Toolbox Items dialog, use the **Browse** button and go to the Quinn-Curtis\DotNet\Lib folder and select QCChart2DNetCF.DLL. OK out of the Choose Toolbox Items dialog. You will now see a **ChartView** control in the tool box list of controls.

- Click on the **ChartView** control in the Toolbox and drop in on the default form.



- When you add a **ChartView** control to a Form, it adds a reference to the DLL that the control is in to the project, and places the default instantiation code in the form's Form.Designer.vb module. Starting with VS 2005, Microsoft has decided to physically separate the designer generated code in a Form.Designer module, while keeping your user written code in a Form.vb module. If you can't see the Form.Designer.vb file in the Solution Explorer, you must click in the **"Show All Files"** icon at the top of the Solution Explorer, and expand the Form.vb entry by clicking on the "+" sign in front Form.vb.

- The default name used for the first instance of the **ChartView** control added to a program is ChartView1. The variable ChartView1 becomes global to the entire Form class. You will see a reference to ChartView1 in all of our example programs. You will find the definition of the ChartView1 variable in the form's Designer.vb module. In our example programs, in the form's user code area, Form.vb we typically assign ChartView1 to a local variable named chartVu, and then customize the graph using the chartVu variable name. You will also need to make sure that you reference **com.quinncurtis.chart2dnetcf**, and **System.Drawing.Drawing2D** in the Imports section of any module that

references **ChartView**. The reference to System.Drawing.Drawing2D is needed just for its definition of the **DashStyle** enumeration which is used by many of the example programs. See the example program UserChartExample1 for a simple example.

```
Imports com.quinncurtis.chart2dnetcf
Imports System.Drawing.Drawing2D

Public Class Form1
    Public Sub InitializeChart()
        Dim chartVu As ChartView = ChartView1
        .
        . ' Chart specific initialization
        .
    End Sub 'InitializeChart

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        InitializeChart()
    End Sub
End Class
```

- We placed all of the chart customization code in the **InitializeChart** method. You can copy this part of the program from our own version of this example, UserChartExample1 Form1.vb. Until this method is called, the **UserChartControl1** appears as an empty shell. We call this method from the Form_Load event. You can easily add the Form_Load event to your program by double clicking on the Form in design mode.

- You should be able to compile the project without error. Run the project and the graph should be visible on the output device, either an actual Windows Mobile device or an emulator. The emulator can take 60 seconds or more to start the program and display the default form.

Adding a ChartView control at runtime

The **ChartView** class can also be instantiated at runtime, without placing it on a form at design time. You can use the default **ChartView** constructor, or one of the special constructors that simplify adding an instance of the control to a form at runtime. See the list of **ChartView** constructors in Chapter 5.

In general you pass the **ChartView** constructor a parent object (Form, Panel or TabbedControl) and a positioning rectangle that defines the position and size of the **ChartView** object. In order to add a **ChartView** control at runtime, the **ChartView** control does NOT need to be added to the Visual Studio Toolbox. You will need to add the QCChart2DNetCF.DLL to the project by right clicking project name in the Solution Explorer and selecting **Add Reference**. Browse to the QCChart2DNetCF.DLL in the Quinn-Curtis\DotNet\lib folder and select it. You will also need to make sure that you **com.quinncurtis.chart2dnetcf**, and **System.Drawing.Drawing2D** in the Imports section of any module that references **ChartView**.

ChartView constructor example (extracted from the example program UserChartExample2, Form1)

In VB, there is no default constructor to invoke the chart initialization. For that reason the chart creation code is placed in the Form_Load event.

```
Imports com.quinncurtis.chart2dnetcf
Imports System.Drawing.Drawing2D


Public Class Form1
    Dim chartView1 As ChartView = Nothing


    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        InstantiateChartView()
        If (chartView1 IsNot Nothing) Then
            InitializeChart()
        End If
    End Sub


    Public Sub InstantiateChartView()
        ' Allow room for a menu in the form
        Dim graphRect As Rectangle = New Rectangle(1, 1, Me.ClientRectangle.Width
- 2, Me.ClientRectangle.Height - 2)


        ' Instantiate chart control
        chartView1 = New ChartView(graphRect, Me)
    End Sub


    Public Sub InitializeChart()
        Dim chartVu As ChartView = chartView1
       .
       .
```

```
        .
    End Sub 'InitializeChart

End Class
```

Adding a UserControl derived from ChartView at runtime

You can derive your own chart class from **ChartView**, and use that control instead of the base **ChartView** class. Right click on the project in the Solution Explorer and select **Add | User Control**. Choose the **UserControl** template. This will add a basic template for a user defined UserControl1 to your project. Go to the **UserControl1.Designer.vb** file and change the inheritance from **UserControl** to **ChartView**. You can then customize the UserControl1 class to produce a specific chart. See the UserControl1.vb code in the UserChartExample3 example program.

In order to add a **ChartView** derived control at runtime, the **ChartView** derived control does NOT need to be added to the Visual Studio Toolbox. You will need to add the QCChart2DNetCF.DLL to the project by right clicking project name in the Solution Explorer and selecting **Add Reference**. Browse to the QCChart2DNetCF.DLL in the Quinn-Curtis\DotNet\lib folder and select it. You will also need to make sure that you reference **com.quinncurtis.chart2dnetcf**, and **System.Drawing.Drawing2D** in the Imports section of any module that references **ChartView**.

ChartView constructor example (extracted from the example program UserChartExample3)

In VB, there is no default constructor to invoke the chart initialization. For that reason the chart creation code is placed in the Form_Load event.

```
 ' From file UserControl1.vb
Public Class UserControl1
'  Chance the inheritance in the UserControl1.Designer.vb file
 '    Inherits ChartView

    Public Sub InitializeChart()

      Dim chartVu As ChartView = Me
        .
        .
        .
    End Sub 'InitializeChart
End Class

' From file Form1.vb
```

```
Public Class Form1
    Dim ChartView1 As UserControl1 = Nothing

    Public Sub InstantiateChartView()
        ' Allow room for a menu in the form
        Dim graphRect As Rectangle = New Rectangle(1, 1, Me.ClientRectangle.Width
- 2, Me.ClientRectangle.Height - 2)

        ' Instantiate chart control and add it to the form
        ChartView1 = New UserControl1()
        ChartView1.Location = graphRect.Location
        ChartView1.Size = graphRect.Size
        ChartView1.PreferredSize = graphRect.Size
        Me.Controls.Add(ChartView1)
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        InstantiateChartView()
        ChartView1.InitializeChart()
    End Sub
End Class
```

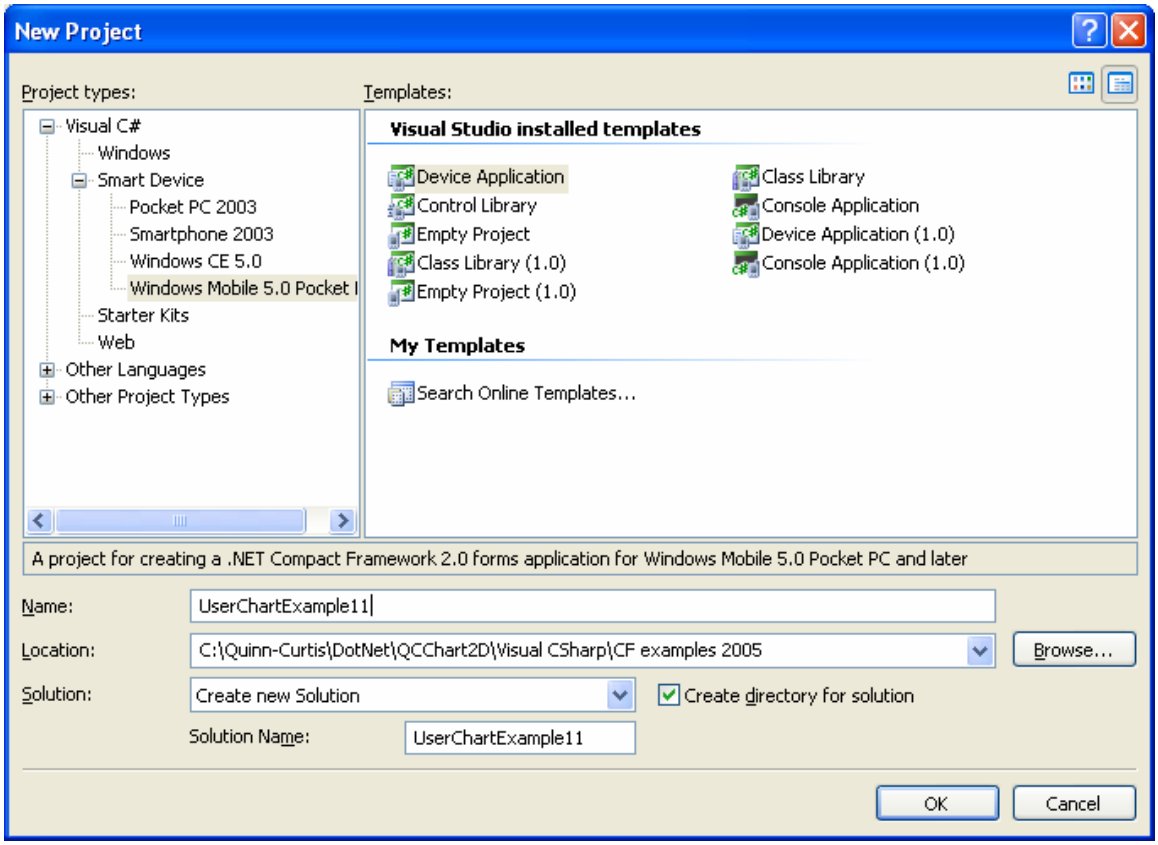
Visual C# for .Net Compact Framework

First, if this is the first **.Net Compact Framework** program you have every created, make a few practice application programs using the Visual Studio defaults. Don't try to add graphics to an application until you are able to create a simple .**Net Compact Framework** applications using the **New Project (File | New | Project)** application wizard.

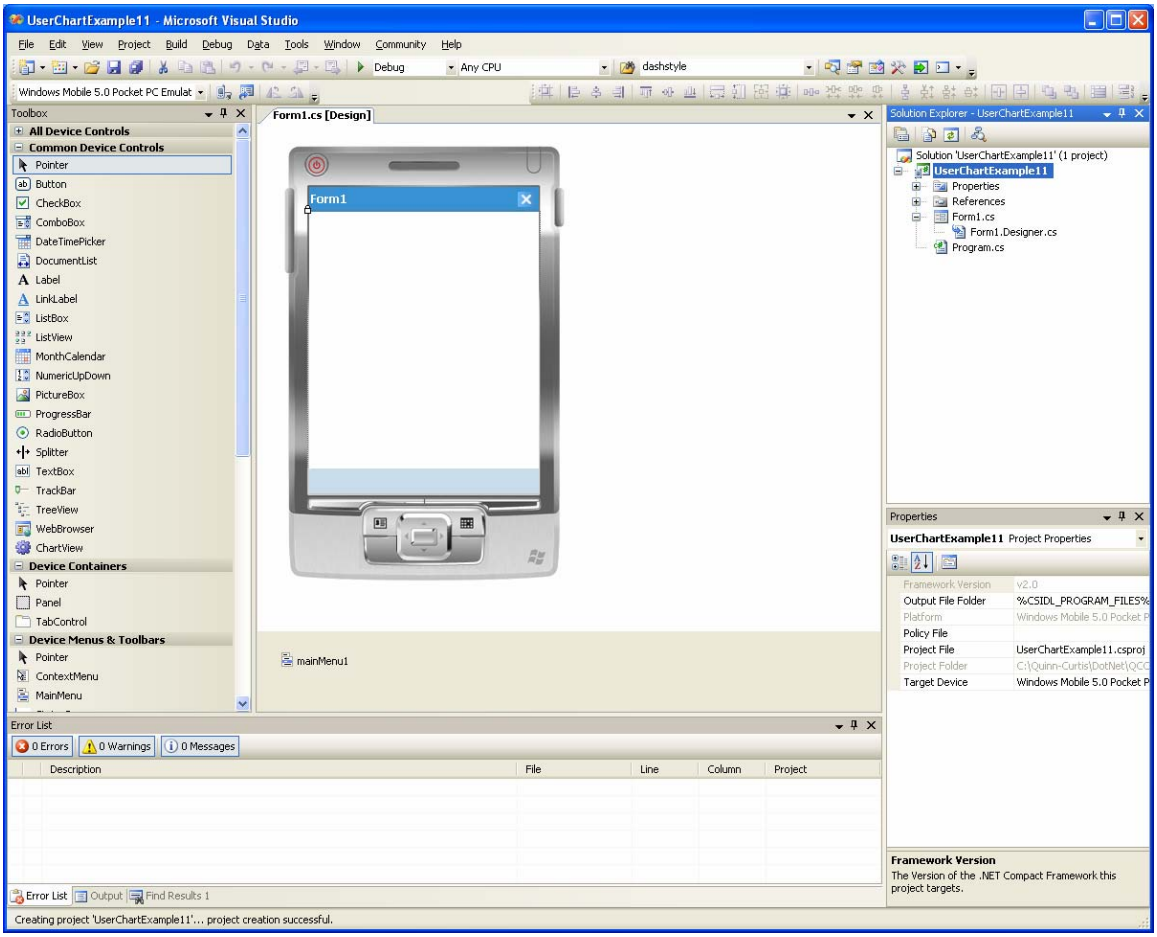
- You start the **New Project** application wizard by selecting **File | New | Project**, bringing up the **New Project** dialog box.

- From this dialog, under Project types, select **Visual C# | Smart Device | Windows Mobile 5.0** folder on the left, and the **Device Application** template on the right. The default Device Application targets a .Net CF 2.0 device. Do **NOT** target a .Net 1.0 device by selecting the Device Application (1.0), since this software is not compatible with .Net 1.0 and 1.1.

- Assign a name to the application in the name box, either the default (**DeviceApplication1**), or your own pick (**UserChartExample11** in the example below). Select a location, which in the cases of our examples is the folder C:\Quinn-Curtis\DotNet\QCChart2D\Visual CSharp\CF examples 2005.



- Click OK and you end up looking at the basic template for a simple application.



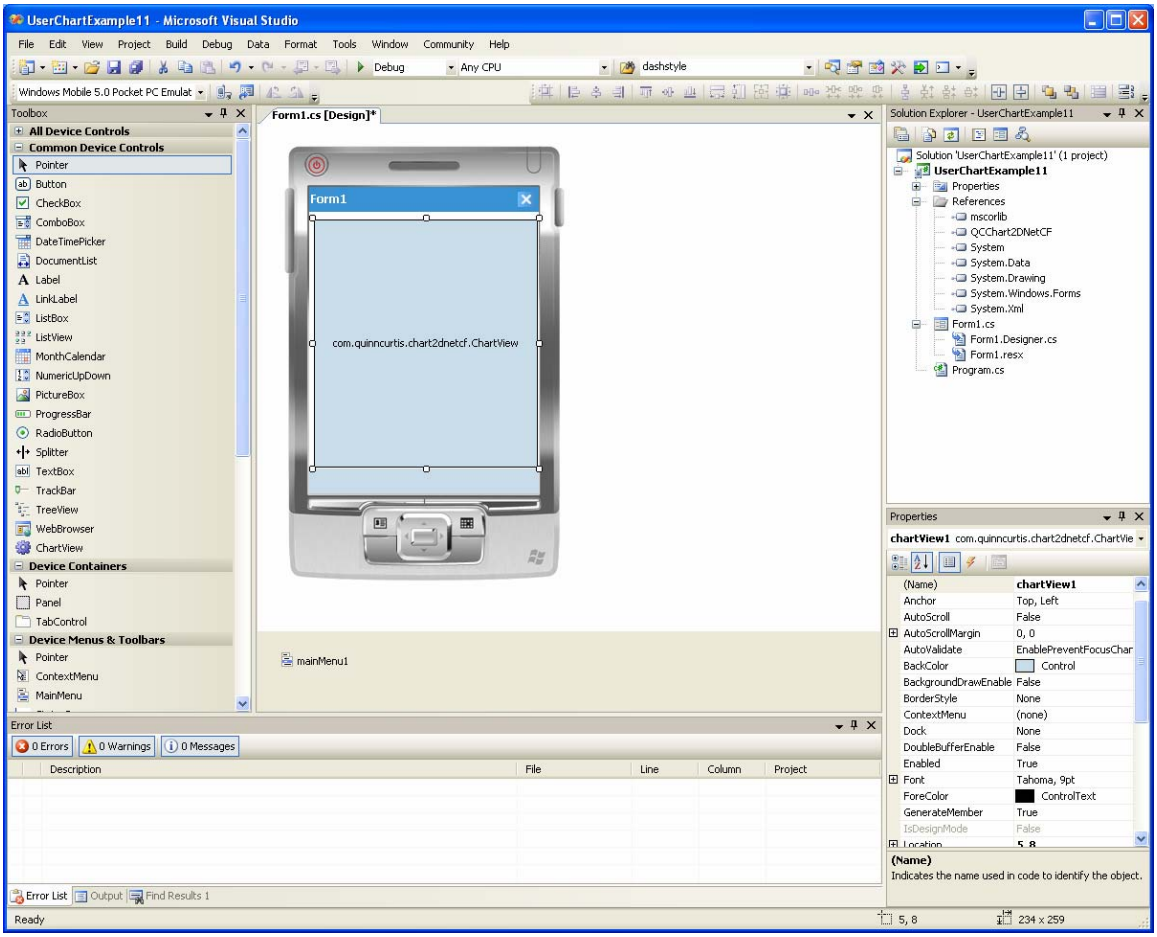
.

The previous steps apply to all **.Net Compact Framework** applications. The following steps are specific to adding **QCChart2D CF** charts to your application. For purposes of this example, the chart will placed in the initial, default form.

Using ChartView with the Visual Studio Design Mode

- **Add the ChartView control to the Toolbox.** If it isn't there already, add the **ChartView** UserControl to the Visual Studio Toolbox. To add the **ChartView** control to the VS Toolbox, right click on the Toolbox and select **Choose Items**. From the Choose Toolbox Items dialog, use the **Browse** button and go to the Quinn-Curtis\DotNet\Lib folder and select QCChart2DNetCF.DLL. OK out of the Choose Toolbox Items dialog. You will now see a **ChartView** control in the tool box list of controls.

- Click on the **ChartView** control in the Toolbox and drop in on the default form.



- When you add a **ChartView** control to a Form, it adds a reference to the DLL that the control is in to the project, and places the default instantiation code in the form's Form.Designer.cs module. Starting with VS 2005, Microsoft has decided to physically separate the designer generated code in a Form.Designer.cs module, while keeping your user written code in a Form.cs module. If you can't see the Form.Designer.cs file in the Solution Explorer, you must click in the "Show All Files" icon at the top of the Solution Explorer, and expand the Form.cs entry by clicking on the "+" sign in front Form.cs.

- The default name used for the first instance of the **ChartView** control added to a program is chartView1. The variable chartView1 becomes global to the entire Form class. You will see a reference to chartView1 in all of our example programs. You will find the definition of the chartView1 variable in the form's Designer.cs module. In our example programs, in the form's user code area, Form.cs, we typically assign chartView1 to a local variable named chartVu, and then customize the graph using the chartVu variable name. You will also need to make sure that you reference **com.quinncurtis.chart2dnetcf**, and **System.Drawing.Drawing2D** in the *using* section of any module that references **ChartView**. The reference to System.Drawing.Drawing2D is needed just for its

definition of the **DashStyle** enumeration used by many of the example programs. See the example program UserChartExample1 for a simple example.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing.Drawing2D;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using com.quinncurtis.chart2dnetcf;

namespace UserChartExample11
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            InitializeChart();
        }

        public void InitializeChart()
        {
            ChartView chartVu = chartView1;
           .
           .
           .
        }
    }
}
```

- We placed all of the chart customization code in the **InitializeChart** method. You can copy this part of the program from our own version of this example, UserChartExample1 Form1.cs. Until this method is called, the **UserChartControl11** appears as an empty shell. We call this method from the the form's constructor. Or you can initialize the graph from the forms Form_Load event, similar to the way it is done using VB.

- You should be able to compile the project without error. Run the project and the graph should be visible on the output device, either an actual Windows Mobile device or an emulator. The eumulator can take 60 seconds or more to start the progam and display the default form.

Adding a ChartView control at runtime

The **ChartView** class can also be instantiated at runtime, without placing it on a form at design time. You can use the default **ChartView** constructor, or one of the special constructors that simplify adding an instance of the control to a form at runtime. See the list of **ChartView** constructors in Chapter 5.

In general, you pass the **ChartView** constructor a parent object (Form, Panel or TabbedControl) and a positioning rectangle that defines the position and size of the **ChartView** object. In order to add a **ChartView** control at runtime, the **ChartView** control does NOT need to be added to the Visual Studio Toolbox. You will need to add the QCChart2DNetCF.DLL to the project by right clicking project name in the Solution Explorer and selecting **Add Reference**. Browse to the QCChart2DNetCF.DLL in the Quinn-Curtis\DotNet\lib folder and select it. You will also need to make sure that you reference **com.quinncurtis.chart2dnetcf**, and **System.Drawing.Drawing2D** in the *using* section of any module that references **ChartView**.

ChartView constructor example (extracted from the example program UserChartExample2, Form1)

The **ChartView** class is instantiated and then initialized, from calls in the forms constructor. Or you can initialize the graph from the forms Form_Load event, similar to the way it is done using VB.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing.Drawing2D;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using com.quinncurtis.chart2dnetcf;

namespace UserChartExample2
{
    public partial class Form1 : Form
    {
        ChartView chartView1 = null;
        public Form1()
        {
            InitializeComponent();
            InstantiateChartView();
            if (chartView1 != null)
```

```
            InitializeChart();
        }

        public void InstantiateChartView()
        {
            // Allow room for a menu in the form
            Rectangle graphRect = new Rectangle(1, 1, this.ClientRectangle.Width -
2, this.ClientRectangle.Height - 2);

            // Instantiate chart control
            chartView1 = new ChartView(graphRect, this);

        }

        public void InitializeChart()
        {
            ChartView chartVu = chartView1;

            .
           .
           .

        }
    }
}
```

Adding a UserControl derived from ChartView at runtime

You can derive your own chart class from **ChartView**, and use that control instead of the base **ChartView** class. Right click on the project in the Solution Explorer and select **Add | User Control**. Choose the **UserControl** template. This will add a basic template for a user defined UserControl1 to your project. Go to the UserControl1.cs and change the inheritance from **UserControl** to **ChartView**. You can then customize the UserControl1 class to produce a specific chart. See the UserControl1.cs code in the UserChartExample3 example program.

In order to add a **ChartView** derived control at runtime, the **ChartView** derived control does NOT need to be added to the Visual Studio Toolbox. You will need to add the QCChart2DNetCF.DLL to the project by right clicking project name in the Solution Explorer and selecting **Add Reference**. Browse to the QCChart2DNetCF.DLL in the Quinn-Curtis\DotNet\lib folder and select it. You will also need to make sure that you reference **com.quinncurtis.chart2dnetcf**, and **System.Drawing.Drawing2D** in the *using* section of any module that references **ChartView**.

ChartView constructor example (extracted from the example program UserChartExample3)

The **UserControl1** class is instantiated and initialized from the forms constructor. Or you can initialize the graph from the forms Form_Load event, similar to the way it is done using VB.

.

‘ From file UserControl1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Text;
using System.Windows.Forms;
using com.quinncurtis.chart2dnetcf;

namespace UserChartExample1
{
    public partial class UserControl1 : ChartView
    {
        public UserControl1()
        {
            InitializeComponent();
        }
        public void InitializeChart()
        {
            ChartView chartVu = this;
           .
           .
           .
      }
    }
}'InitializeChart
```

‘ From file Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing.Drawing2D;
```

```
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using com.quinncurtis.chart2dnetcf;

namespace UserChartExample3
{
    public partial class Form1 : Form
    {
        UserControl1 chartView1 = null;
        public Form1()
        {
            InitializeComponent();
            InstantiateChartView();
            if (chartView1 != null)
                chartView1.InitializeChart();
        }

        public void InstantiateChartView()
        {
            // Allow room for a menu in the form
            Rectangle graphRect = new Rectangle(1, 1, this.ClientRectangle.Width -
2, this.ClientRectangle.Height - 2);

            // Instantiate chart control
            chartView1 = new UserControl1();
            chartView1.Location = graphRect.Location;
            chartView1.Size = graphRect.Size;
            chartView1.PreferredSize = graphRect.Size;
            this.Controls.Add(chartView1);
        }

    }
}
```

24. Frequently Asked Questions

FAQs

1. Is the **QCChart2D CF** for the **.Net Compact Framework** software compatible with the **QCChart2D** for .Net software?

2. Why doesn't your QCChart2D CF software appear on the .Net Compact Framework toolbox.

3. How do you create a chart with multiple coordinate systems and axes?

4. 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?

5. How do you zoom charts that use multiple coordinate systems?

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 the **QCChart2D CF** software to generate JPG, DIB, BMP, GIF, EMF, WMF, etc. 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 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. Are you going to add additional real-time routines to the **QCChart2D CF** for the **.Net Compact Framework** library?

22. Are you going to add 3D routines to the **QCChart2D CF for the .Net Compact Framework** library?

1. Is the **QCChart2D CF** for the **.Net Compact Framework** software compatible with the **QCChart2D** for .Net software?

Yes, assuming you do not use any of the features that are not supported in **QCChart2D CF**, such as rotated text, printer output and output of image files.

2. Why doesn't your QCChart2D CF software appear on the .Net Compact Framework toolbox. And why can't I add it there.

Starting with Version 1.8 of the QCChart2D software, the ChartView UserControl can be added to the Toobox. To add the **ChartView** control to the VS Toolbox, right click on the Toolbox and select **Choose Items**. From the Choose Toolbox Items dialog, use the **Browse** button and go to the Quinn-Curtis\DotNet\Lib folder and select QCChart2DNetCF.DLL. OK out of the Choose Toolbox Items dialog. You will now see a **ChartView** control in the tool box list of controls.

3. 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 hav 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 miniumum 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 miniumum 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(ChartObj.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 MultiAxes 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#]

```
CartesianCoordinates pTransform1;
CartesianCoordinates pTransform2;
CartesianCoordinates pTransform3;
CartesianCoordinates pTransform4;
CartesianCoordinates pTransform5;
.
.  // Initialize datasets, coordinate system ranges
.
// The x-scale range for pTransform1 to pTransform5 are all the same, 0 - 100
// The y-scale range for pTransform1 to pTransform5 are all different

// The plotting area for each pTransform is indentical, leaving a large open
// to the left for extra axes.
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) ;

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);
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);

Font axisTitleFont = new Font("Microsoft Sans Serif", 10, FontStyle.Bold);
AxisTitle xaxistitle = new AxisTitle( xAxis, axisTitleFont, "Event Partition");
chartVu.AddChartObject(xaxistitle);

Grid xgrid = new Grid(xAxis, yAxis1,ChartObj.X_AXIS, ChartObj.GRID_MAJOR);
chartVu.AddChartObject(xgrid);

SimpleLinePlot thePlot1 = new SimpleLinePlot(pTransform1, Dataset1, attrib1);
chartVu.AddChartObject(thePlot1);

SimpleLinePlot thePlot2 = new SimpleLinePlot(pTransform2, Dataset2, attrib2);
chartVu.AddChartObject(thePlot2);

SimpleLinePlot thePlot3 = new SimpleLinePlot(pTransform3, Dataset3, attrib3);
chartVu.AddChartObject(thePlot3);

SimpleLinePlot thePlot4 = new SimpleLinePlot(pTransform4, Dataset4, attrib4);
chartVu.AddChartObject(thePlot4);

SimpleLinePlot thePlot5 = new SimpleLinePlot(pTransform5, Dataset5, attrib5);
chartVu.AddChartObject(thePlot5);
```

[VB]

```
Dim pTransform1 As CartesianCoordinates
Dim pTransform2 As CartesianCoordinates
Dim pTransform3 As CartesianCoordinates
Dim pTransform4 As CartesianCoordinates
Dim pTransform5 As CartesianCoordinates
Dim xAxis As LinearAxis
```

```
Dim yAxis1 As LinearAxis
Dim yAxis2 As LinearAxis
Dim yAxis3 As LinearAxis
Dim yAxis4 As LinearAxis
Dim yAxis5 As LinearAxis
.
.  ' Initialize datasets, coordinate system ranges
.
' The x-scale range for pTransform1 to pTransform5 are all the same, 0 - 100
' The y-scale range for pTransform1 to pTransform5 are all different

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 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)
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)

Dim axisTitleFont As New Font("Microsoft Sans Serif", 10, FontStyle.Bold)
Dim xaxistitle As New AxisTitle(xAxis, axisTitleFont, "Event Partition")
chartVu.AddChartObject(xaxistitle)

Dim xgrid As New Grid(xAxis, yAxis1, ChartObj.X_AXIS, ChartObj.GRID_MAJOR)
```

```
chartVu.AddChartObject(xgrid)

Dim thePlot1 As New SimpleLinePlot(pTransform1, Dataset1, attrib1)
chartVu.AddChartObject(thePlot1)

Dim thePlot2 As New SimpleLinePlot(pTransform2, Dataset2, attrib2)
chartVu.AddChartObject(thePlot2)

Dim thePlot3 As New SimpleLinePlot(pTransform3, Dataset3, attrib3)
chartVu.AddChartObject(thePlot3)

Dim thePlot4 As New SimpleLinePlot(pTransform4, Dataset4, attrib4)
chartVu.AddChartObject(thePlot4)

Dim thePlot5 As New SimpleLinePlot(pTransform5, Dataset5, attrib5)
chartVu.AddChartObject(thePlot5)
```

4. **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 **CustomChartDataCursor** class, creates a new **Marker** object and **NumericLabel** object each time a mouse button clicked.

[C#]

```
Marker amarker = new   Marker(GetChartObjScale(), MARKER_BOX,
           nearestPoint.GetX(), nearestPoint.GetY(),  10.0, PHYS_POS);
chartVu.AddChartObject(amarker);
rNumericLabelCntr += 1.0;
// Add a numeric label the identifies the marker
pointLabel = new NumericLabel(GetChartObjScale(),
           textCoordsFont, rNumericLabelCntr, nearestPoint.GetX(),
           nearestPoint.GetY(), PHYS_POS, DECIMALFORMAT, 0);
// Nudge text to the right and up so that it does not write over marker
```

```
pointLabel.SetTextNudge(5,-5);
chartVu.AddChartObject(pointLabel);
chartVu.UpdateDraw();
```

[VB]

```
Dim amarker As New Marker(GetChartObjScale(), MARKER_BOX, nearestPoint.GetX(),
           nearestPoint.GetY(), 10.0, PHYS_POS)
chartview.AddChartObject(amarker)
rNumericLabelCntr += 1.0
' Add a numeric label the identifies the marker
pointLabel = New NumericLabel(GetChartObjScale(), textCoordsFont,
           rNumericLabelCntr, nearestPoint.GetX(), nearestPoint.GetY(),
           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()
```

5. 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.

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.Windows.Forms.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 SpectrumAnalyzer, DataLogger, DynPieChart and ScrollingMixedPlot 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 **Control** display buffer. The **ChartView** 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?

Implementing drill down or tool tips consists of three major parts:

a. Trapping a mouse event and determining the mouse cursor position in device and physical coordinates.

b. Identifying the chart object that intersects the mouse event.

c. 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)
```

Some of the example programs that include tool tips include LineFill, Multiline, LinePlotSegments, StackedLineChart, Logarithmic, SimpleBarChart, GroupBarPlotChart, DoubleBarPlot, OpeningScreen, OHLCFinPlot and LabeledPieChart.

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   = chartView1;

// 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 = ChartView1

' 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   = chartView1;
// 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 datset
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 = ChartView1
' 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 datset
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 **Control**. 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** or **PolarCoordinates**) for each chart, and to position the plot area 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. The example below was extracted from the OHLCFinPlot class.

[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 the QCChart2D CF software to generate JPG, DIB, BMP, GIF, EMF, WMF, etc. files?

The **.Net Compact Framework** does not support the imaging routines needed to save charts as bitmap image files or metafiles. If you need to save charts as image files or metafiles you should be using the related product, **QCChart2D** for .Net.

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 ordered 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  = chartView1;
.
.
.
```

```
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.SetZOder(150); // Grid is drawn after ChartPlot objects
                     // which have default z-value of 50
chartVu.AddChartObject(grid2);
```

[Visual Basic]

```
Dim chartVu As ChartView = ChartView1
.
.
.

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.SetZOder(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?

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 **Control** for its canvas and the scroll bars will sit on top of this, not side by side. The example program 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.

As of this writing there seems to be serious problems with the .Net scrollbars, largely due to the circumstance that they do not support trapping MouseDown and MouseUp

events. Since **.Net Compact Framework** devices are relatively slow, you do not want to place CPU intensive code in the HScrollBar1.ValueChanged (or VScrollBar1.ValueChanged) event handler, as we do in the regular .Net version of **QCChart2D**. This is because that event is triggered for every pixel change in the value of the scrollbar, causing it to get bogged down when you try and move the scrollbar. A better technique is to only process changes in the graph when the MouseUp event is triggered by the scrollbar, signifying that the scroll operation is complete. But as we said, the MouseUp events don't work. So we cheat, when a scrollbar value changes, we start a timer and only update the graph one second after the initial change. This gives the user time to complete the scrollbar movement and does not bog things down with multiple updates of the graph for intermediate values.

[C#]

```
public void UpdateXScaleAndAxes(int index)
{
    int startindex = index;
    pTransform1.SetScaleStartX( (double) startindex);
    pTransform1.SetScaleStopX( (double) (startindex + 50));
    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_ValueChanged(object sender, System.EventArgs e)
{
    hScrollActive = true;
    this.timer1.Enabled = true;
```

```
}

private void vScrollBar1_ValueChanged(object sender, System.EventArgs e)
{
    vScrollActive = true;
    this.timer1.Enabled = true;
}

private void timer1_Tick(object sender, System.EventArgs e)
{
    if (hScrollActive)
    {
        UpdateXScaleAndAxes(hScrollBar1.Value);
        hScrollActive = false;
    }
    if (vScrollActive)
    {
        UpdateYScaleAndAxes( vScrollBar1.Value);
        vScrollActive = false;
    }
    this.timer1.Enabled = false;
}
```

[Visual Basic]

```
Public Sub UpdateXScaleAndAxes(ByVal index As Integer)
   Dim startindex As Integer = index
   pTransform1.SetScaleStartX(CDbl(startindex))
   pTransform1.SetScaleStopX(CDbl(startindex + 50))
   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 HScrollBar1_Scroll(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles HScrollBar1.ValueChanged
   hScrollActive = True
   Me.Timer1.Enabled = True
End Sub

Private Sub VScrollBar1_Scroll(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles VScrollBar1.ValueChanged
   vScrollActive = True
   Me.Timer1.Enabled = True
End Sub

Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
   If (hScrollActive) Then
      UpdateXScaleAndAxes(HScrollBar1.Value)
       hScrollActive = False
   End If
   If (vScrollActive) Then
       UpdateYScaleAndAxes(VScrollBar1.Value)
         vScrollActive = False
   End If
   Me.Timer1.Enabled = False
End Sub
```

There are other examples of Form components interacting with charts. In the OHLCFinPlot example a **Scrollbar** controls the time axis of a stock market OHLC chart. The MultiAxes example uses **Button** objects to select the x-axis range.

19. I am trying to plot 100,000 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 mobile device chart image will have 300-600 pixels as the horizontal resolution of the plotting area. This would imply that in the 100K data point example above, every horizontal pixel would represent 1.6K to 3.3K 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 100K data points and compress them down to 500 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 a 1000 point compressed dataset, where there are at least two data points per pixel of horizontal resolution, will look just like the 100K point dataset, only display a hundred 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 = 100000;
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 = 100000
Dim RawDataset As TimeSimpleDataset = new _
      TimeSimpleDataset("Raw", xtimedata,  ydata,nNumPnts)
Dim compressXmode As Integer = ChartObj.DATACOMRESS_AVERAGE
Dim compressYmode As 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. Are you going to add additional real-time routines to the QCChart2D CF for the .Net Compact Framework library?

Yes, see the ***Real-Time Graphics Tools for .Net CF*** http://www.quinn-curtis.com/QCRTGraphCFProdPage.htm

22. Are you going to add 3D routines to the QCChart2D CF for the .Net Compact Framework library?

Probably not. 3D Graphics are popular in presentation charting applications because management types think that 3D charts in their PowerPoint presentations make the content more believable. In most implementations of 3D charting, the use of the 3rd

dimension adds nothing to, and often obscures, the information content of the chart. It also makes real-time updates and interactive zooming slower and less precise.

Also, we are not looking for new ways to make the **.Net Compact Framework** slower than it already is.

INDEX