QCTAChart - Technical Analysis Charting Tools for .Net

Contact Information

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

General Information: info@quinn-curtis.com
Sales: sales@quinn-curtis.com

Technical Support Forum

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

Revision Date 1/022/2015 Rev. 2.4

Documentation and Software Copyright **Quinn-Curtis, Inc. 2015**

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

IMPORTANT-READ CAREFULLY: This Software End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Quinn-Curtis, Inc. for the Quinn-Curtis, Inc. SOFTWARE identified above, which includes all Quinn-Curtis, Inc. ***.Net*** 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.

Table of Contents

1. Introduction....1
Financial Data Sources....5
Technical Analysis Charting Tools for .Net Dependencies....6
Directory Structure of QCTAChart for .Net....6
(*** Critical Note ***) Running the Example Programs....8
Chapter Summary....8
Tutorials....9
Customer Support....9
2. QCTAChart and Technical Analysis....13
Primary Chart plotting options....15
Technical Indicator overlays for the Primary chart....34
Secondary windows technical indicators....43
Other Chart Features....59
3. Class Architecture of the Technical Analysis Charting Tools for .Net Class Library 69
Charting Tools for .Net Class Summary....69
Technical Analysis Charting Tools for .Net Class Summary....70
4. QCChart2D for .Net 81
QCChart2D for .Net Class Summary....81
Chart Window Classes....82
Data Classes....82
Scale Classes....83
Coordinate Transform Classes....83
Auto-Scaling Classes....85
Chart Object Classes....85
Mouse Interaction Classes....110
File and Printer Rendering Classes....111
Miscellaneous Utility Classes....111
5. Configuring QCTAChart Datasources....115
Getting Started with a Data Source....116
6. Display Stock Data in the Primary Chart....141
7. Secondary Chart Options....163
8. Financial Chart Objects 195
9. Point and Figure Charts....223
10. Renko Charts....243
11. File and Printer Rendering Classes....257
Printing a Chart....257
Capturing the Chart as a Buffered Image....263
....267
12. Regionalization for non-USA Markets....269
13. Using Technical Analysis Charting Tools for .Net to Create Windows Applications....283
(*** Critical Note ***) Running the Example Programs....283
Visual C# for .Net....283
....291
Visual Basic for .Net....291

1. Introduction

Technical Analysis of stocks, bonds, commodities and other securities, is the art of predicting future price movements (for the security) based on the study of past price and volume movements. Starting with historical Open-high-low-close-volume data for a given security, technical analysis consists of applying one or more functions to that data, creating an indicator which predicts the future trend of the security, whether it be up, down or flat. Buy and sell actions for the security are then decided based on the indicator.

That's it in a nutshell. The defining postulate of technical analysis is that the historical market action of a security takes into account everything publicly known about the security. There is no need for the technical trader to study a stocks balance sheet, income statement, cash flow, or recent product announcements (the bread and butter of a competing strategy known as *fundamental analysis*), because all of these things are already factored in the stocks price. This assumes that the security trades in a free market, where the price action is controlled by thousands, or even millions of independent buyers and sellers, all operating using their own criteria. Don't expect technical analysis to work for securities which have an extremely small float (penny stocks for example), or are largely controlled by a single trading entity (stock trading scams where the where the price is manipulated by a coordinated network of brokers, buyers and sellers – think the *Wolf of Wall Street*), or in stock exchanges run by countries which to not believe in free market economics and are willing to manipulate a stock price to their own end.

Technical analysis does not require that the practitioner use charting. It is straightforward to use technical analysis to produce buy, hold and sell signals, without ever looking at a chart. The computer allows a skilled programmer to automate the analysis of historical stock data, and produce buy and sell signals using technical analysis without the user every seeing a Open-high-low-close chart, or a technical indicator chart. Traders involved in computer-driven trading will execute trades based on a computer generated buy or sell signal. In this case the program is processing technical indicators derived from the fundamental Open-high-low-close-volume data, and looking for specific buy and sell signals.

Usually the technical analysis programmer tasked with automating an existing application has to make a decision about the amount of programming he wants to do. Does he purchase an application package that implements standard technical analysis charts and then go about defining the charts using some sort of menu driven interface or wizard. This is probably the most expensive in terms of up front costs, and on-going subscription fees, and the least flexible, but the cheapest in development costs since a programmer does not have to get involved creating the displays. Another choice is to use a general purpose spreadsheet package with charting capability to record, calculate, and display the charts. This is probably a good choice if your charting needs are simple, and you are prepared to write complicated formulas as spreadsheet entries, and your data input is not automated. Another choice is writing the software from scratch, using a charting toolkit like our ***QCChart2D*** software as the base, and creating custom technical analysis charts using the primitives in the toolkit. This is cheaper up front, but may be expensive in terms of development costs. Often times the third option is the only one available because the end-user has some unique requirement that the pre-packaged software can't handle, hence everything needs to programmed from scratch.

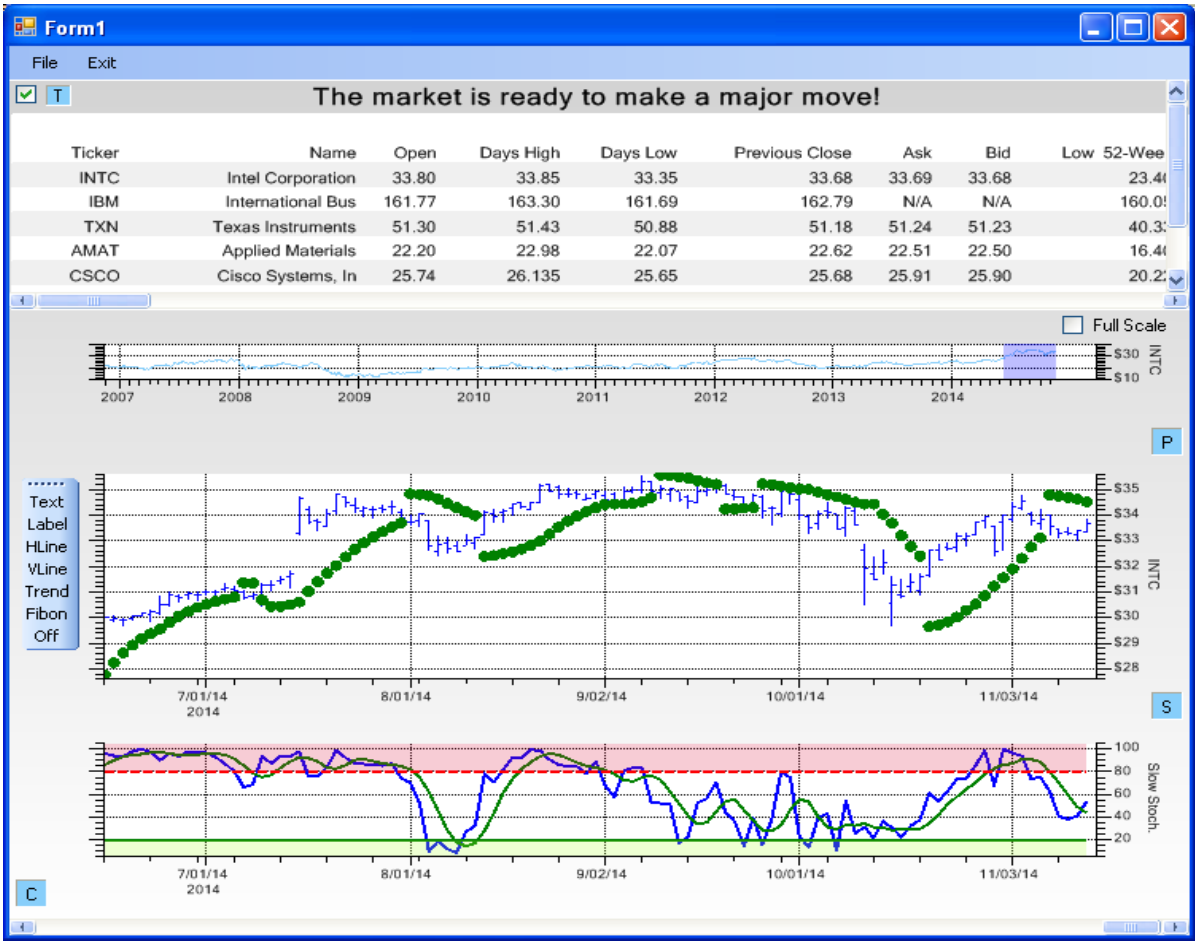
QCTAChart for Technical Analysis

The **QCTAChart** software package is for those who do want to use charting as part of their decision making process. We have created a library of technical analysis routines that represents an intermediate solution. Our technical analysis software still requires an intermediate level programmer, but it does not

require advanced knowledge of technical analysis or of charting. Typically, a user will choose a stock, and a time frame to analyze. The historical stock data will be displayed as an Open-high-low-close chart (or candlestick chart). The user will have the option a applying a selection of time-invariant transfer functions to the data, producing additional charts which will indicate buy or sell signals on inspection. In some cases, an indicator will overlay the existing Open-high-low-close chart, and in others they will be displayed in a synchronized window under the main Open-high-low-close chart.

QCTAChart is a template that integrate the **QCChart2D** charting software with tables, data structures and specialized rendering routines used for the static and dynamic display of technical analysis charts. The chart template uses pre-programmed classes that create, manage and display the technical analysis charts and tables. The template can be further customized using methods and properties. The programmer can customize the plot objects created in the template, allowing tremendous flexibility in the look of the technical charts.

A QCTAChart has four major major plotting areas: a current financial data window, a zoom window, a primary chart window, and secondary indicator windows. A typical QCTAChart is shown below. The top window, which is all text, is the current financial data window. The small chart under that is the zoom control for the next chart, which is the primary technical chart window. The smaller windows under that the are the secondary technical indicator windows.



This chart combines a current data table, a zoom window, a Primary candlestick chart displaying a Parabolic SAR overlay, and a Secondary chart displaying a Slow Stochastic indicator chart.

Primary Chart Window

The Primary chart window can display the OHLC stock data for up to three stocks using the following plot options:

- Classic Open-high-low-close
- Candlestick
- Simple line plot of close data
- Mountain (or filled) line plot of close data
- Equi-volume candlestick plot
- Open-High-Low-Close Bar plot
- Point and Figure plot
- Renko plot
- Time frame (starting date/time and frame extent is controlled using a dedicated zoom control window
- Data values for the plot, at the intersection of a mouse controlled vertical cursor with the plot, are displayed above each chart window

You can also overlay the Primary chart using one or more of the following technical indicators

- Simple moving averages
- Exponential moving averages
- Moving Average Bands
- Bollinger Bands
- Parabolic SAR

Some the other options associated with the Primary chart are:

- Dynamic auto-scaling to displayed data
- Compare up to three different stocks
- Linear, Logarithmic, and Normalized y-axis scale
- Synchronize scrolling (panning) and zooming of all chart windows
- Financial Chart objects which can be dropped into a chart

    Horizontal and Vertical data makers

Fibonacci overlay

Labels for Annotation

Trend lines

Arrows

Secondary Chart Window

The Secondary window can display the following technical indicators:

- Average Directional Indicator
- Momentum\\
- Rate of Change (ROC)
- Relative Strength (RSI)
- Stochastic (Fast and Slow)
- Williams %R
- Moving Average Convergence/Divergence (MACD)
- Volume charts

Integrated Data Table

A scrollable table (at the top) which contains current financial data for a user-defined portfolio of stocks. The user can click on a row and automatically have all of the charts updated to reflect technical indicators for the selected stock.

Integrated Dialog Boxes

There are Primary Chart, Secondary Chart, Data Table, and General Chart characteristic dialog boxes which are invoked with small buttons on the main view page. The end-user can customize which stocks and technical indicators are displayed. Plot attributes (plot type, line and fill colors, line thickness), text fonts and text sizes are all adjustable.

Serialization

The Primary Chart dialog box has options to Save and Restore the complete setup of a chart which has been customized by the end-user.

Financial Data Sources

Paid Data Sources

A technical analysis charting package means little if you do not have a source of historical data for the securities you want to analyze. There are many sources of paid historical data. If you work for a large financial company, you may have access to computer feeds from Thompson Reuters, Bloomberg, Wall Street Journal, Metastock and many others. You can get historical data in file form, and on-line. Since data in file form must be constantly updated in order to keep it current, the best source is going to be on-line. Paid, online, data sources are going to cost anywhere from hundreds, to tens of thousands of dollars per year, depending on how many financial instruments you have access to (there are tens of thousands), how many historical data sets a day you expect to retrieve, and the frequency of the historical data (end-of-day (EOD) data, minute by minute, 5-second, or even tick by tick. This software is designed to work with any frequency of data. What we don't have at this time are drivers for these paid, historical data feeds. Should you provide us with a specification, for the URL query, and the return data format, we can probably create a custom data source module for you which will let you read data from that data source for little or no money.

Free Data Sources

There are several, free, real-time URL sources of historical stock data. These are Yahoo Finance, Google Finance, and Quandl. The use of historical data from these sources is completely free. What you can't do is download historical data from them (it's free after all) and then sell, or give it away to someone else. That would be a copyright violation. But, you can use the data in your stock trading program(s) to make decisions about when to buy or sell stocks, without paying them any royalties. Most of the historical stock data is end-of-day (EOD) data. It can go back twenty or thirty years for some stocks. Yahoo and Google also offer intra-day data (stock prices throughout a trading day) going back up to twenty days. The frequency of the data (whether or not it is minute by minute data, or some multiple thereof, varies.

There are some issues with the free historical data sources. In most cases, once you go back ten or fifteen years, the Open-high-low-close-volume format of data degrades to just daily Close-Volume data, and finally to just the close price for the security. So technical indicators which depend on Open, High, Low and Volume values can't be used for the time range in those case. Also, historical data is dependent on stock splits, and some data sources normalize the stock data for stock splits and others don't. When the free data sources do adjust the OHLC data for stock splits, it often takes a month or two after the split occurs. Until that time, charts of the stock will show a drastic price drop on the date of the stock split, since that is what the data will show. During that update interval, before the historical data has been normalized for the stock splits, technical analysis will not be practical. This was more of an issue fifteen years ago, where high growth stocks split every one or two years, not so much now. Apple (AAPL) recently split though so watch out for that. Paid data sources such as Thompson Reuters are always going to have better, more update to date data, with complete Open-high-low-close data and data always properly normalized for stock splits.

Yahoo Finance

Yahoo Finance is a web site that provides current and historical financial information on USA and many non-USA markets. This includes quotes on stocks, bonds, commodities and futures. In addition to EOD data, Yahoo Finance also has an API for retrieving intraday data for stocks . Yahoo Finance is probably the most popular site for market data in the USA. Our software can access Yahoo Finance for historical data feeds of EOD data, and historical data feeds up to 20 days of intraday data..

Google Finance

Google Finance, launched in 2006, was meant to be a competitor to Yahoo Finance. It offered much the same as Yahoo in terms of current and historical data for stocks, bonds, commodities and stock futures. Unfortunately, Google announced in 2013 that they were discontinuing most of Google Finance. However, the current, and historical data quote service, still seems to be working. So use it at your own risk, and only with a reliable backup data source. In addition to EOD data, Google Finance also has an API for retrieving intraday stock data. Our software can access Google Finance for historical data feeds of EOD data, and historical data feeds up to 20 days of intraday data..

Quandl

Quandl is a relatively new data source, started a couple of years ago. The site offers access to several million financial, economic and social datasets, including EOD data for a large number of USA and non-USA stocks. They also have a constantly updated, and standardized collection of financial ratio data for the same stocks. Unlike Google and Yahoo, Quandl does not have a data source for intraday stock data. Quandl seems to be committed to providing free access to EOD financial data. You will need to register with them if you want to make heavy usage of their data. In return they give you a key, or taken, which is used in the accessing the Quandl URL address. Our software can access Quandl for historical data feeds, and for current financial ratio data feeds.

Metastock

Metastock has been in the business of supplying market data, for a subscription fee, for 30 years. Originally they supplied data on media using a proprietary binary format. Starting with Metastock 12, the shifted to supplying data to their paying customers using online methods only. Then, starting with Metastock 13, they reversed that policy and added support for an easy to read CSV file structure for storing stock OHLCV data in a file on a local computer. Many other companies also use the Metastock CSV file format for their own data, so it has become a standard for OHLCV data in file format. Our software can read Metastock CSV historical data files, and if the standard Metastock first line header is part of the file, automatically adjust to any of Metastock column formats.

The **QCTAChart** is a template that integrate the **QCChart2D** charting software with tables, data structures and specialized rendering routines used for the static and dynamic display of technical analysis charts. The chart template uses pre-programmed classes that create, manage and display the technical analysis charts and tables. The template can be further customized using methods and properties. The programmers can customize the plot objects created in the template, allowing tremendous flexibility in the look of the technical charts.

Technical Analysis Charting Tools for .Net Dependencies

The **QCTAChart** class library builds on the **QCChart2D** software package. You must use the version of QCChart2D which ships with the QCTAChart software. It uses the classes found in that software and standard classes that ship with the Microsoft .Net API. It also makes use of the open source Newtonsoft.Json library for some JSON parsing (see http://james.newtonking.com/json). No other software is required.

Directory Structure of QCTAChart for *.Net*

The **QCTAChart** class library uses the standard directory structure also used by the **QCChart2D, QCSPCChart, and QCRTGraphics** software. It adds the **QCTAChart** directory structure under the Quinn-Curtis\DotNet folder. For a list of the folders specific to **QCChart2D**, see the manual for **QCChart2D**, QCChart2DNetManual.pdf.

Drive:

Quinn-Curtis\ - Root directory

DotNet\ - Quinn-Curtis .Net based products directory

Docs\ - Quinn-Curtis .Net related documentation directory

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

QCChart2D\ - QCChart2D examples for C# and VB – This directory contains many example programs for C# and VB specific to the QCChart2D charting software, but not specific to the QCTAChart software

QCTAChart\ - QCTAChart examples for C# and VB

Visual CSharp\ - C# specific directory

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

Examples\ - C# examples directory

FinChartObjectsExample – demonstrates the use of the trend line, Fibonacci object, horizontal and vertical cursors, and annotation labels

IntradayDatasource – demonstrates how to display intraday data obtained from Yahoo and Google data sources.

PointAndFigureExample – demonstrates a Point and Figure chart, and Renko chart, which obtains data from the Yahoo-base data source.

QuandlDataSourceExample – display a technical analysis charting using Quandl as the data source.

YahooDataSourceExample - display a technical analysis charting using Yahoo as the data source.

GoogleDataSourceExample - display a technical analysis charting using Google as the data source..

MetastockFileSourceExample - display a technical analysis charting using a Metastock CSV file as the data source.

TAChartApplication1 – the example used in the Tutorial in Chapter 13

Visual Basic\ - VB specific code

Examples\ - VB examples

Same as the C# examples above

(*** Critical Note ***) Running the Example Programs

The example programs for **QCTAChart** software are supplied in complete source. In order to save space, they have not been pre-compiled which means that many of the intermediate object files needed to view the main form are not present. This means that **FinChartView** derived control will not be visible on the main Form if you attempt to view the main form before the project has been compiled. The default state for all of the example projects should be the Start Page. Before you do view any other file or form, do a build of the project. This will cause the intermediate files to be built. If you attempt to view the main Form before building the project, Visual Studio sometimes decides that the **FinChartView** control placed on the main form does not exist and deletes it from the project.

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

30-Day Trial Version

The trial version of **QCTAChart** is downloaded in a file named Trial_QCTAChartR24x. The 30-day trial version stops working 30 days after the initial download. The trial version includes a version message in the upper left corner of the graph window that cannot be removed.

Developer Version

The developer version of **QCTAChart** is downloaded in a file with a name similar to NETTADEVR2x4x561x1.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. *Save your download link*.

Chapter Summary

The remaining chapters of this book discuss the **QCTAChart** package designed to run on any hardware that has a .Net runtime 3.5 installed on it.

Chapter 2 presents a summary of the technical analysis charts that can be created using the software.

Chapter 3 presents the overall class architecture of the **QCTAChart** and summarizes all of the classes found in the software.

Chapter 4 summarizes the important **QCChart2D** classes that you must be familiar with in order to customize advanced features of the **QCTAChart** software.

Chapter 5 describes the data source classes and how they can be customized for your application..

Chapter 6 describes how to create the Primary stock chart in the **FinChartView** class, and how to add technical indicator overlays.

Chapter 7 describes how to add one or more Secondary technical analysis charts underneath the Primary stock chart.

Chapter 8 describes how to add Financial Chart Objects (trend lines, annotation text, horizontal and vertical data markers, and Fibonacci overlays to the Primarystock chart.

Chapter 9 describes how to create Point and Figure charts using the software.

Chapter 10 describes how to create Renko charts using the software

Chapter 11 describes how to print the charts, and save them to image files.

Chapter 12 describes how to regionalize the software for non-USA English markets.

Chapter 13 is a tutorial that describes how to use **QCTAChart** to create Windows applications using Visual Studio .Net, Visual C# and Visual Basic.

Tutorials

Chapter 13 is a tutorial that describes how to get started with the **QCTAChart** charting software.

Customer Support

Use our forums at http://www.quinn-curtis.com/ForumFrame.htm for customer support. Please, do not post questions on the forum unless you are familiar with this manual and have run the examples programs provided. We try to answer most questions by referring to the manual, or to existing example programs. We will always attempt to answer any question that you may post, but be prepared that we may ask you to create, and send to us, a simple example program. The program should reproduce the problem with no, or minimal interaction, from the user. You should strip out of any code not directly associated with reproducing the problem. You can either your own example or a modified version of one of our own examples.

2. QCTAChart and Technical Analysis

Since you are a programmer looking to write a Technical Analysis software package, it is assumed that you are already familiar with the basic assumptions. So we jump right in to the different chart types and indicators available in this software toolkit. The main chart types, and indicators were introduced in Chapter 1 .

Primary Chart plotting options

- Classic Open-high-low-close plot
- Candlestick
- Simple line plot of close values
- Mountain (or filled) close plot
- Equi-volume candlestick plot
- Point and Figure chart
- Renko chart
- Time frame (starting date/time and frame extent is controlled using a dedicated zoom control window
- Data values for the plot, at the intersection of a mouse controlled vertical cursor with the plot, are displayed above each chart window

Technical Indicators which can overlay the OHLC plot in the Primary chart

- Simple moving averages
- Exponential moving averages
- Moving Average Bands
- Bollinger Bands
- Parabolic SAR

Other options for the Primary chart

- Dynamic auto-scaling to displayed data
- Compare up to three different stocks
- Linear, Logarithmic, and Normalized y-axis scale
- Synchronize scrolling (panning) and zooming of all chart windows

Technical Indicators which can can be displayed in separate, secondary windows

- Average Directional Indicator
- Momentum
- Rate of Change (ROC)
- Relative Strength (RSI)
- Stochastic (Fast and Slow)
- Williams %R
- Moving Average Convergence/Divergence (MACD)
- Volume charts

Other Related Charting Features

- An event-based coordinate system suitable use for financial data where data has an irregular time stamps because weekends, holidays, and other discontinuities.
- Financial Chart objects which can be dropped into a chart
- Horizontal and Vertical data markers
- Fibonacci overlay
- Labels for Annotation
- Trend lines
- Arrows

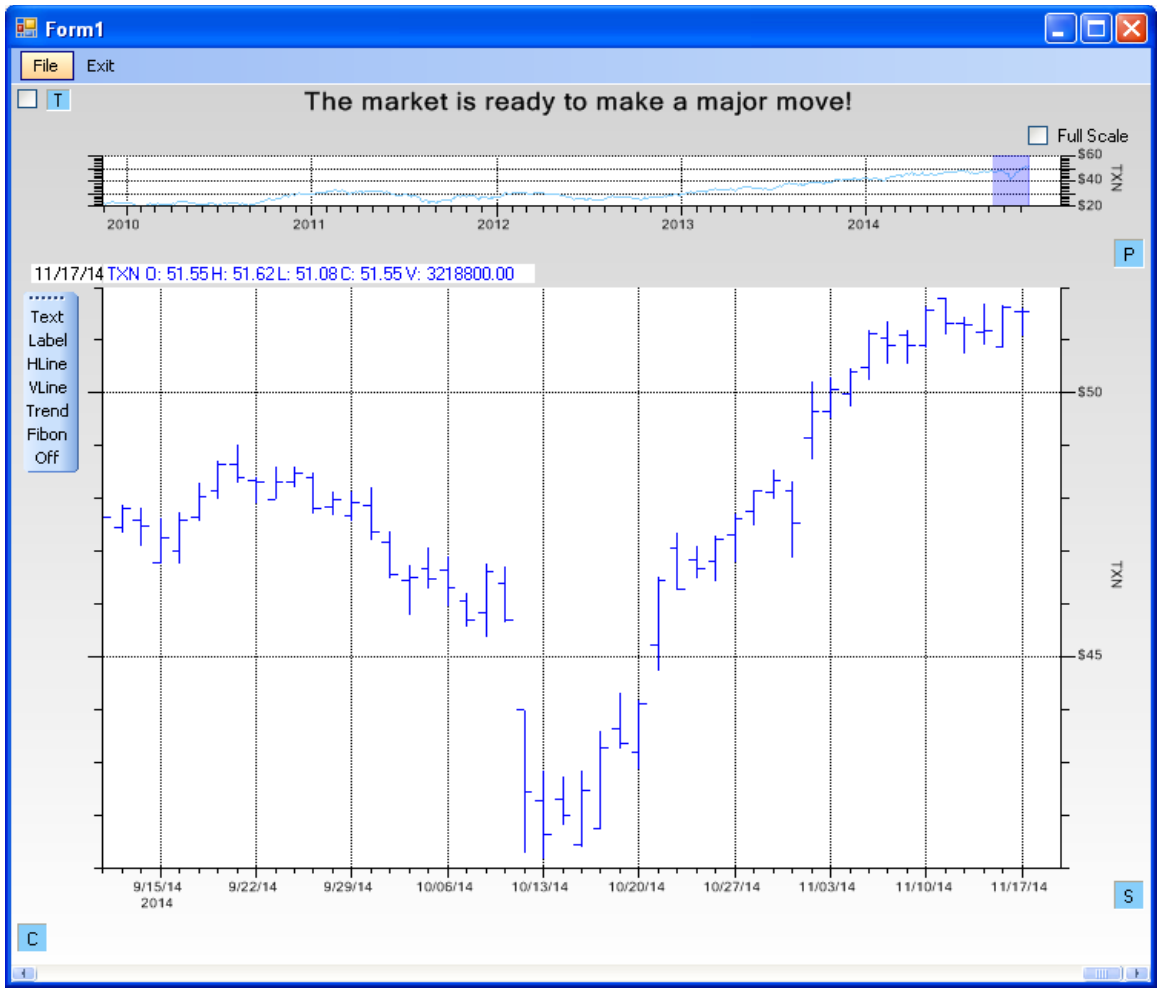
A scrollable table (at the top) which contains current financial data for a user-defined portfolio of stocks. The user can click on a row and automatically have all of the charts updated to reflect technical indicators for the selected

stock.

Primary Chart plotting options

The Primary Chart displays a plot of one to three stocks in one of several different formats: simple line plot, OHLC plot, Candlestick plot, Mountain Plot, OHLC Bar plot, Equi-volume Candlestick plot. The y-axis can be configured for a simple linear scale, a normalized linear scale, and a logarithmic scale. The user can also enable one or more technical indicator overlays, directly on top of the plot. These overlay indicators include: simple moving averages, exponential moving averages, moving average bands, Bollinger bands, and the Parabolic SAR. Also, a single stock can be displayed as a Point and Figure plot, or a Renko plot. Chapter 6 describes the programming of the Primary Chart display in detail.

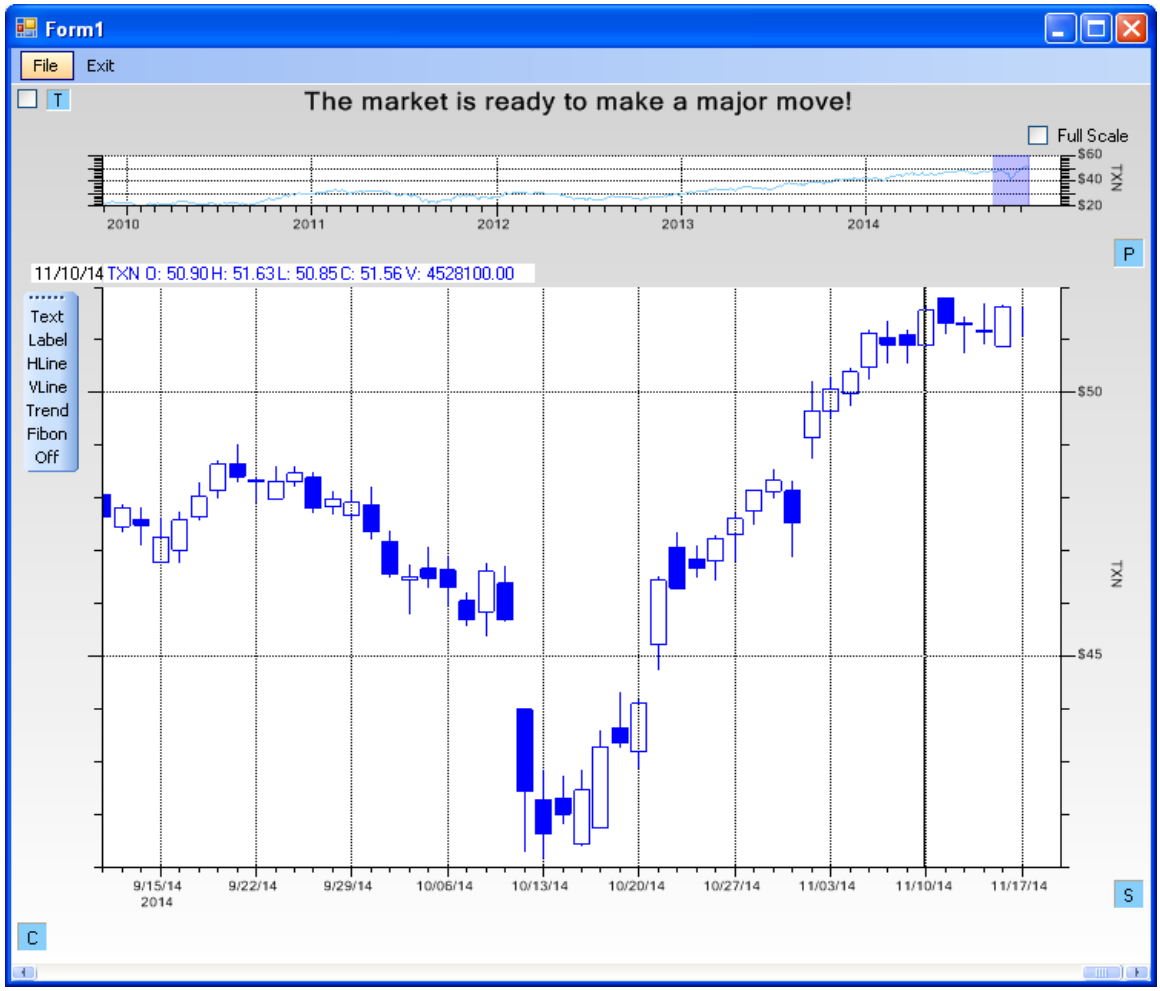
Classic Open-high-low-close plot



The classic OHLC chart used in technical analysis. Data can be zoomed and panned using the upper zoom window, or the scrollbar at the bottom.

The **OHLCPlot** class displays stock market data in an open-high-low-close format common in financial technical analysis. Every event 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. It is most useful when used to display data consisting of 10 to 50 events. Otherwise the vertical lines get too close together and it degenerates into a big smear with most of the information content lost.

Candlestick Plot



The classic Candlestick chart used in technical analysis, where the candlestick is filled if the close for that time period is lower than the open.

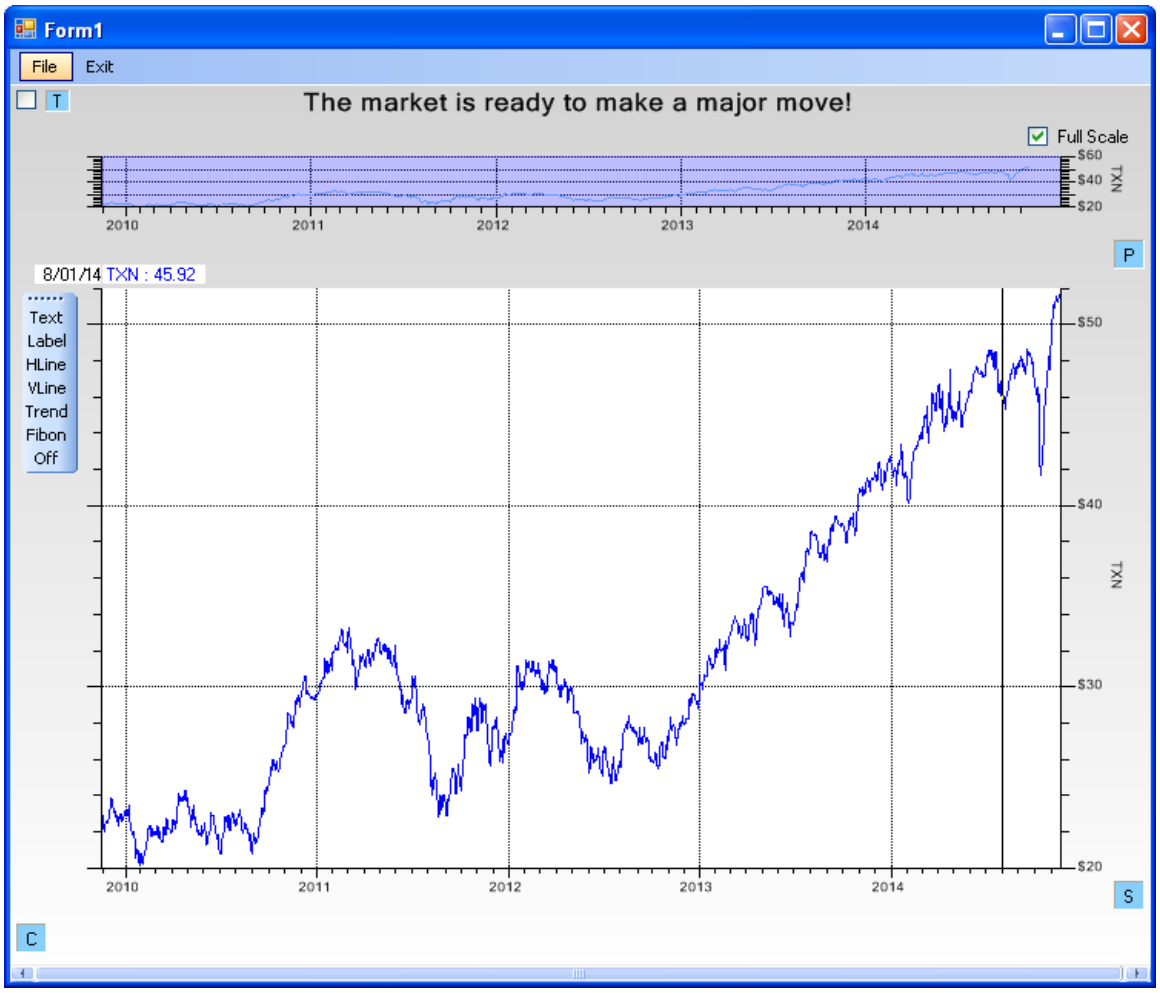
The candlestick plot displays stock market data in an open-high-low-close format common in financial technical analysis. Every item of the plot is represented by a vertical line representing High and Low values, overlapped by a box representing the Open and Close values. If the Close value is greater than the Open value for a particular candlestick, the box is filled, otherwise it is unfilled. Like the OHLC plot, the candlestick plot is best used when there are between 10 and 50 event items in the current time frame, else the details of the candlesticks will be too small to see.

OHLC Bar plot

This OHLC Bar plot is a variant of the candlestick plot. The three colors of the OHLC bar define the four OHLC values. A dot in the middle means that the Open is higher than the Close.

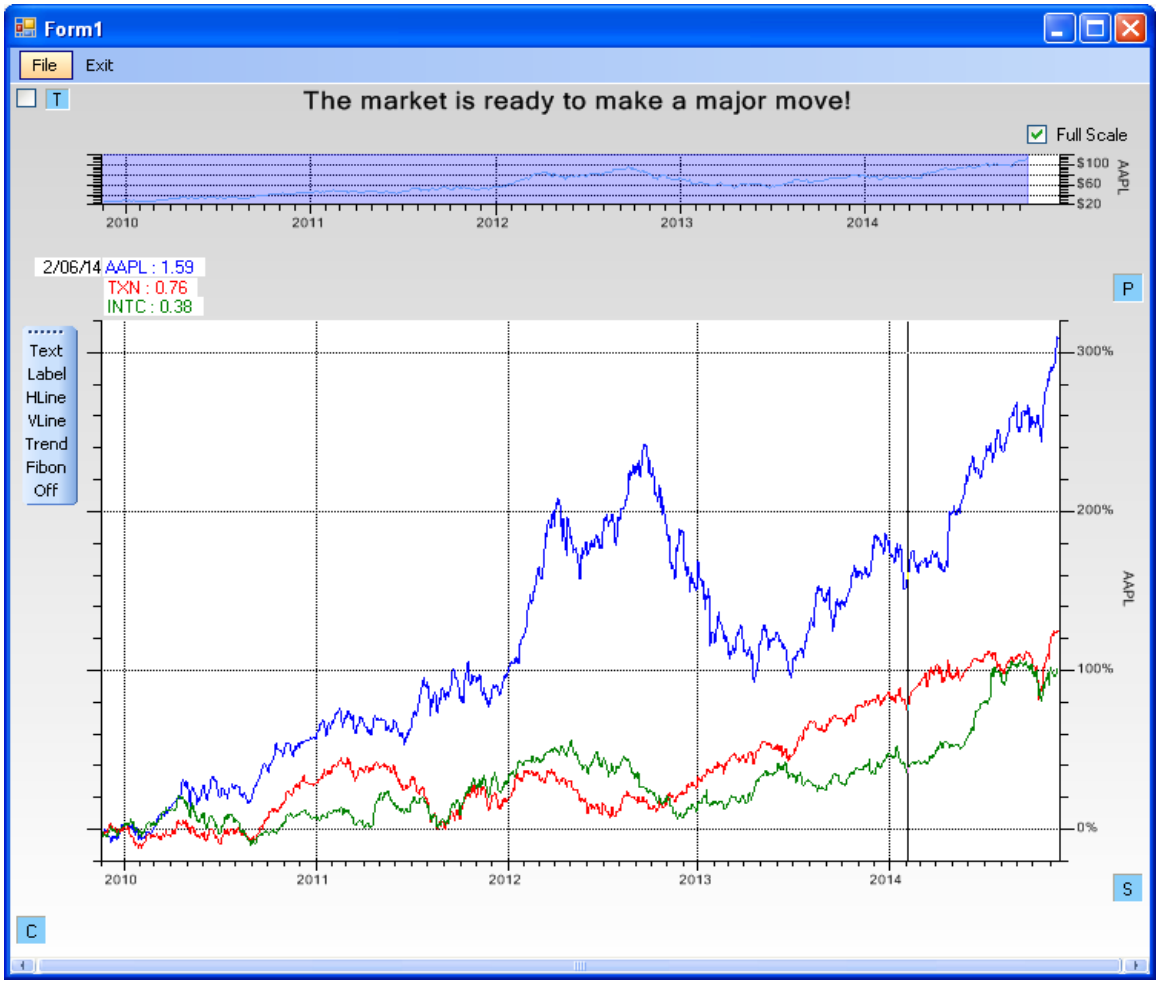
The OHLC bar plot is a variant of the candlestick plot. Every item of the plot is represented by a multicolored bar representing Open-High-Low-Close values. If the Open value is greater than the Close value for a particular bar, the center box has a dot placed in the center, otherwise no dot is present. Like the OHLC plot, the OHLC bar plot is best used when there are between 10 and 50 event items in the current time frame, else the details will be too small to see.

Simple line plot of close values



When you have more then 50 or so data items, it is best to use the line plot type displaying the closing values.

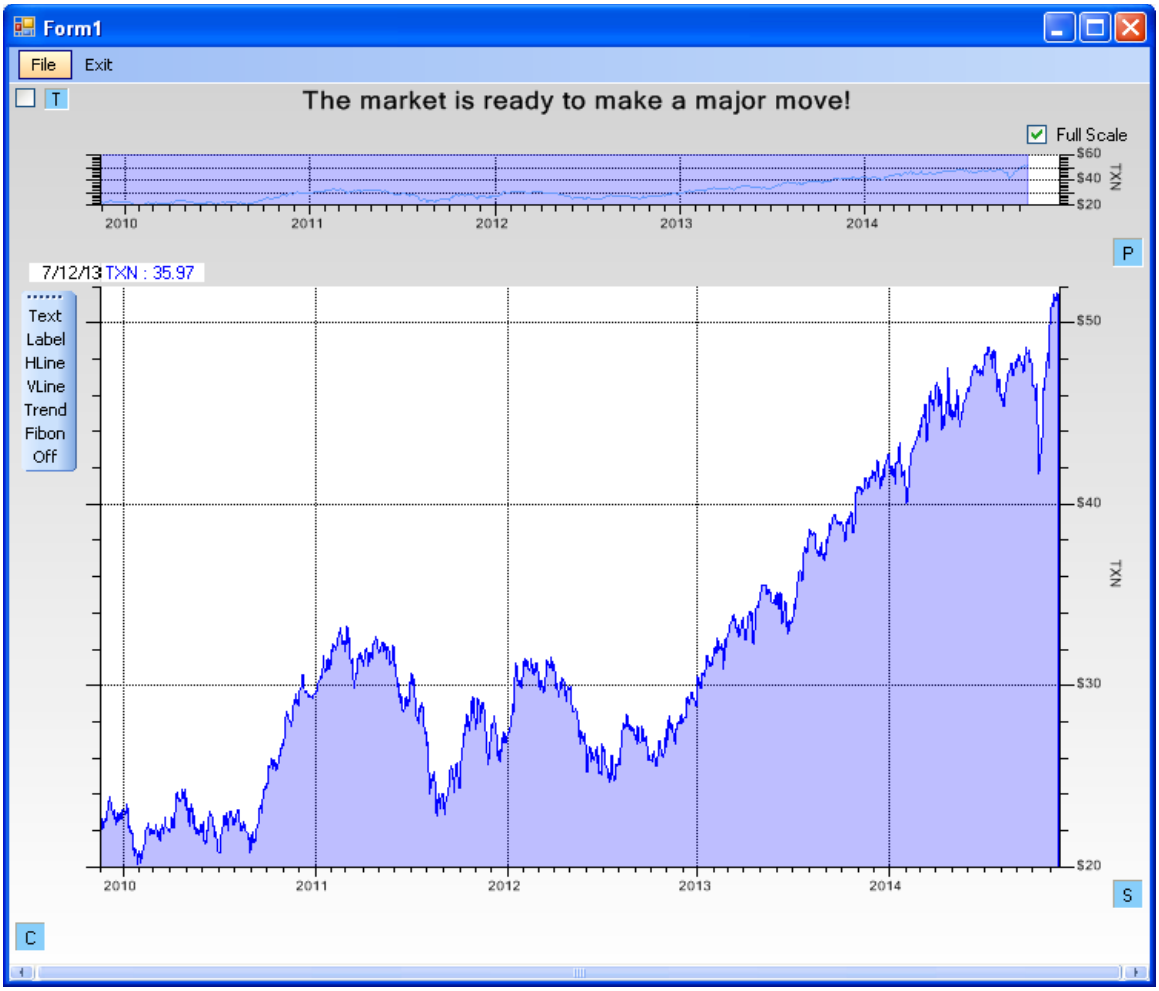
For charts which extend across a large number of event items, a line plot of just the close values is the most legible.



When you compare two or three stock, using a normalize y-axis scale, it is best to use the line plot type.

When you compare two or more stocks of different ranges, use the Normalized y-scale. It normalizes the compared stocks to 0% at the beginning of the full-range time frame. In the example above, the blue line represents the normalized stock price of Apple, which increased 200% to 300% more than its peers.

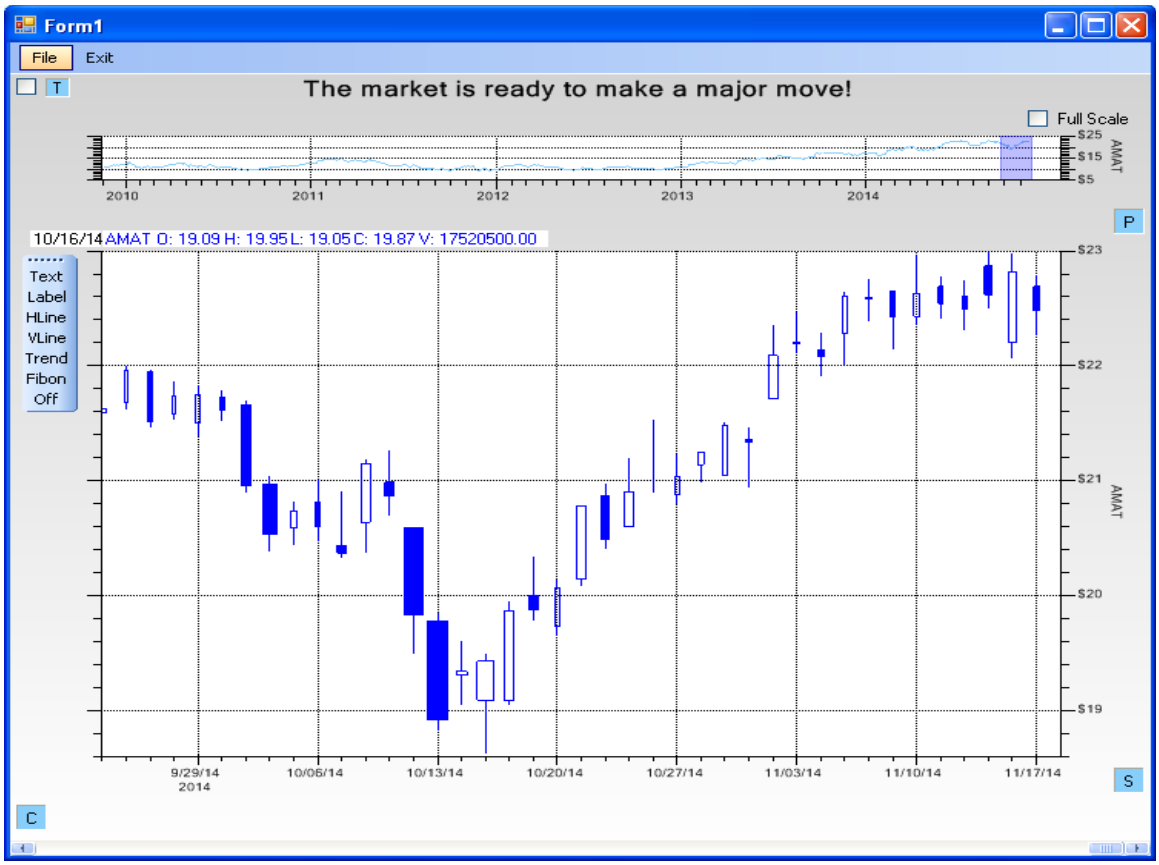
Mountain (or filled) close plot



A filled version of the line plot is called a Mountain plot.

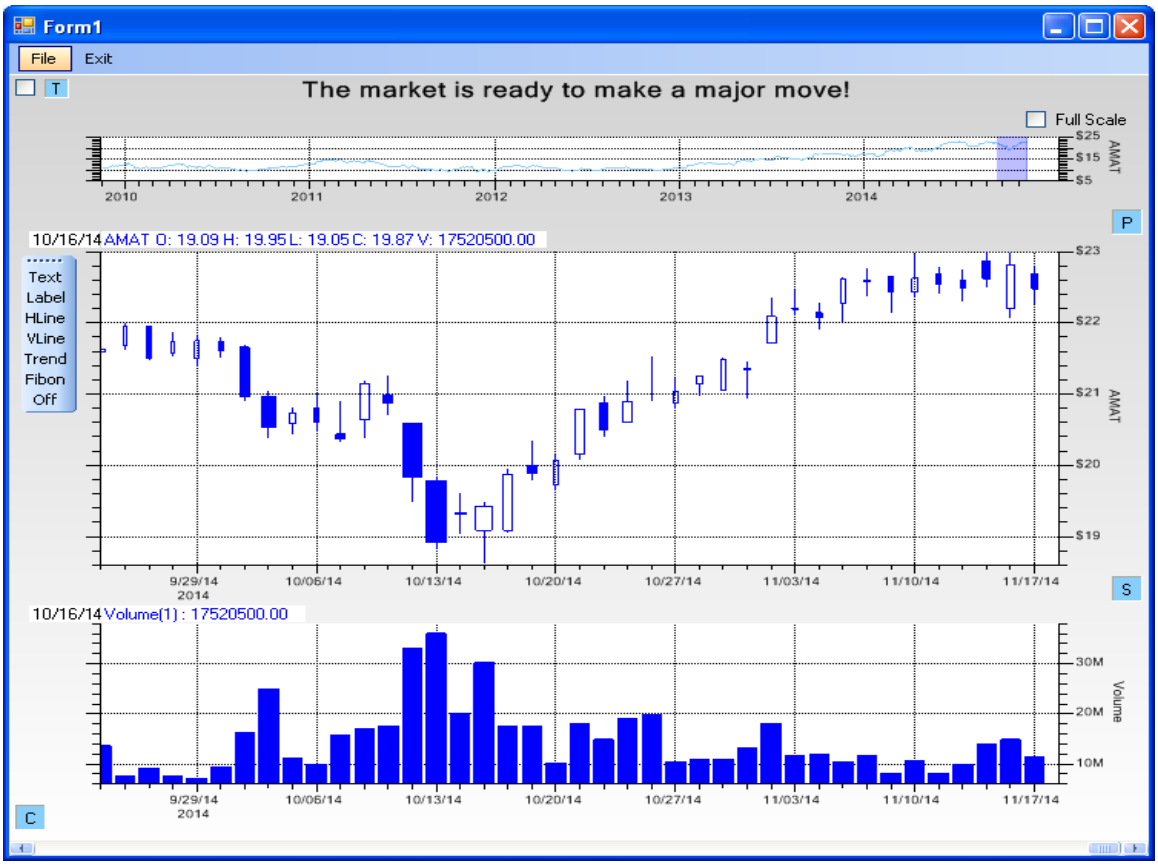
A Mountain plot is just a line plot, with the area under the line filled down to the x-axis.

Equi-volume candlestick plot



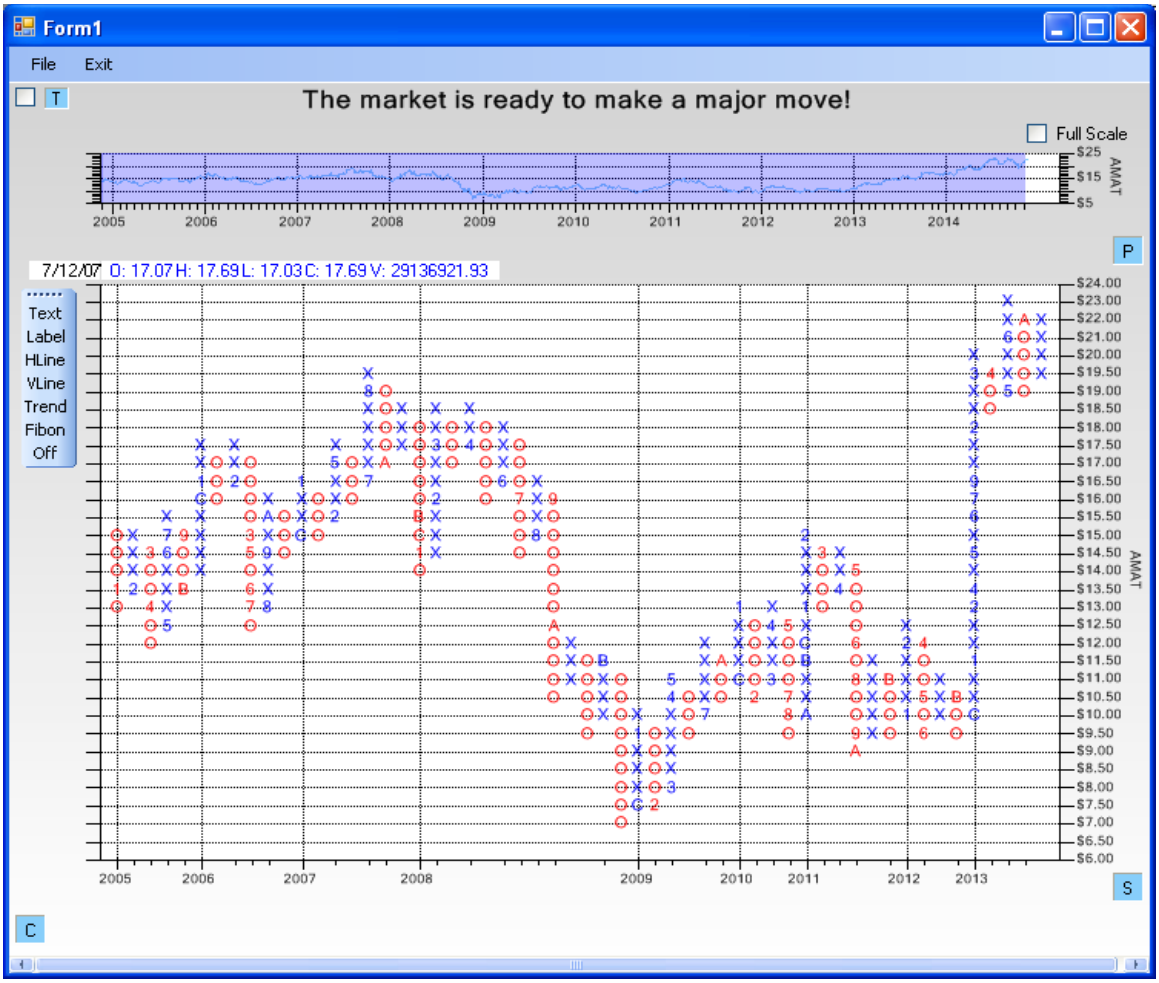
The Equi-volume plot is a Candlestick plot which uses a variable width for the candlestick box. The item width is set proportional to the volume for that time period, adding another dimension to the chart.

There is a variant of the Candlestick plot called the Equi-volume plot. The variant is that relative volume information for each candlestick is encoded in the candlestick width. We do not implement the traditional Equi-volume, because it uses a very irregular x-axis which will not sync up with our other technical indicator plots. Instead, we look at the data in the current view of the chart, and assign the maximum candlestick width to the maximum Volume value (from the OHLCV data) in the current view. That volume is assigned a width of 1.0, and all other candlesticks are assigned widths less than one, equal to the ratio of Volume/(Maximum Volume). When you look at a Equi-volume plot, you will immediately see which candlesticks occurred on days with the highest volume, since they will be the widest. In the example above, the day with the highest volume are 10/13/2014. If you plot an actual volume plot underneath, you can see the relationship between candlestick width, and volume.



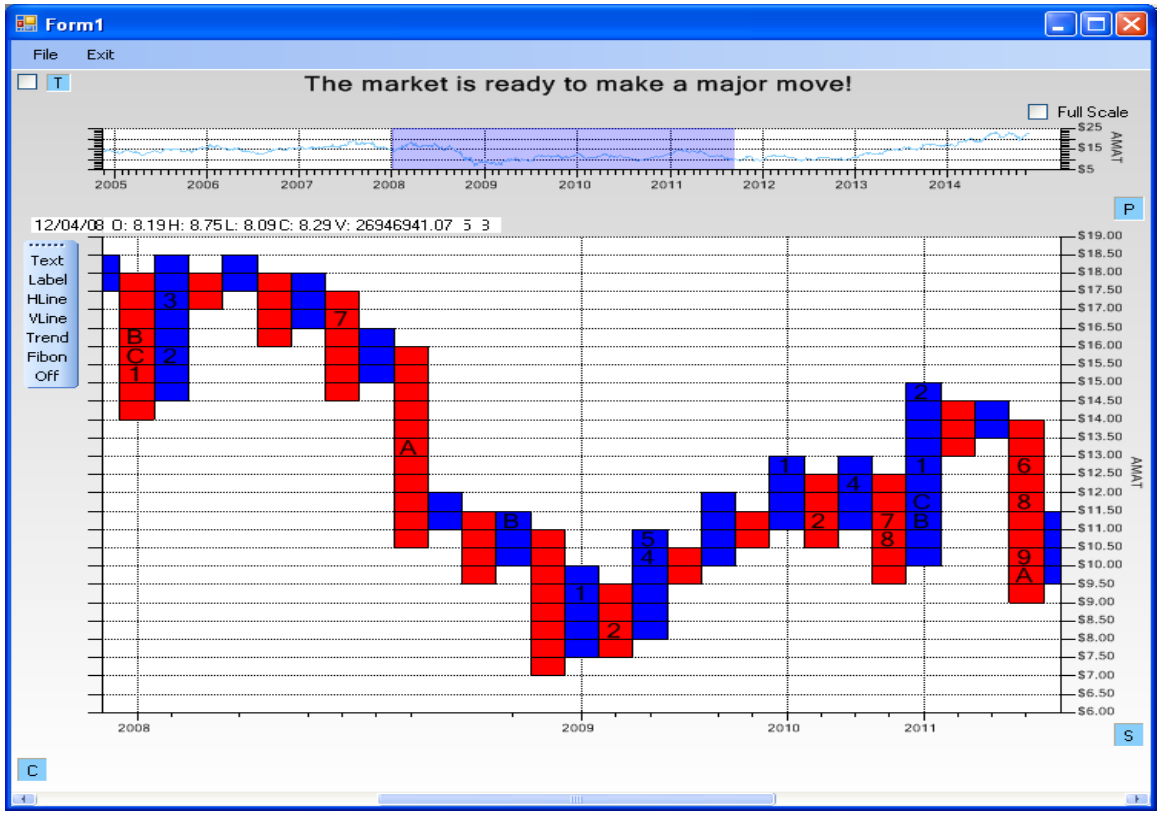
In the picture above, the fattest candlestick items correspond to the time periods where the trading volume was the highest.

Point and Figure Charts



Point and Figure charts do not plot price against time as other techniques do. Instead it plots price against changes in direction by plotting a column of Xs as the price rises and a column of Os as the price falls.

Point and Figure plots have been used in technical analysis for more than 100 years. It is unique in that it does not plot price against time as other techniques do. Instead it plots price against changes in direction by plotting a column of Xs as the price rises and a column of Os as the price falls. As long the stock price is increasing, and does not backtrack by more than a multiple (usually 3) of the box size, the price increase is displayed as an increasing vertical column of Xs, one X for each time the stock price breaks through the top of a box price level. Once the trend reverses more than a multiple of the box value, the column increments to the right, and changes over to a column of 0's, which are plotted down as long as the stock price continues to drop, without any significant reversals. Many technicians like it because it filters out much of the normal up and down noise in the stock data, and makes it very easy to identify trends up or down. As in the example above, it compresses the time frame, so that many years of data (eight+ in the example above) can be displayed without crowding.

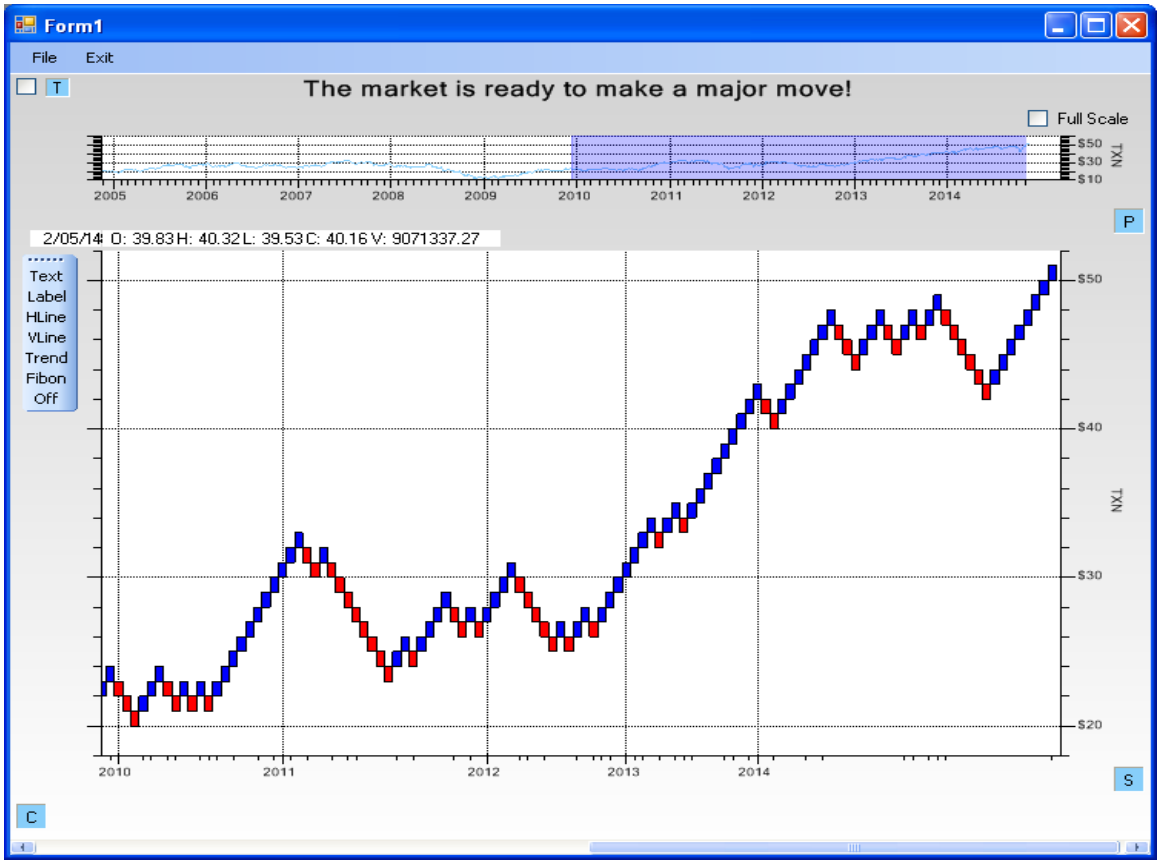


While not a traditional Point and Figure chart, the floating bar version displays the same information, using alternating bar colors instead of the Xs and Os.

A good book on the subject is *Point and Figure Charting* by Thomas J. Dorsey.

More detailed description of our Point and Figure chart implementation is found in Chapter 9, *Point and Figure Charts*.

Renko Charts

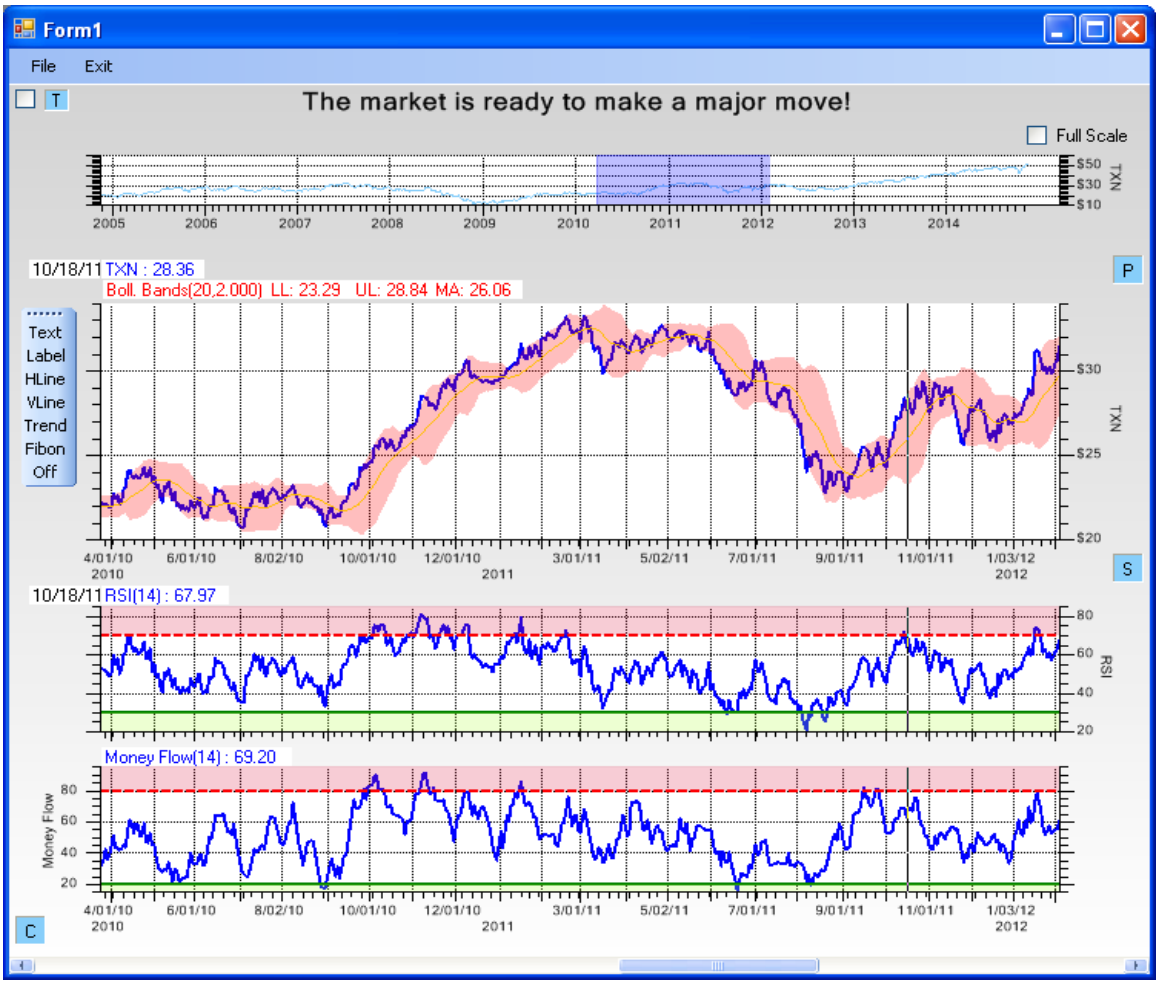


Renko charts do not plot price against time as other techniques do. Instead it plots price against changes in direction by plotting a column of Xs as the price rises and a column of Os as the price falls.

Renko charts are similar to Point and Figure charts, in that they do not plot price against time as other techniques do. Instead it plots price against changes in direction by plotting filled boxes (called *bricks* in Renko terminology) as the price rises and unfilled boxes as the price falls. As long the stock price is increasing, and does not backtrack by more than the brick size size, the price increase is displayed as an rising diagonal of bricks. Each time the price rises enough to warrant a new brick, a new column is started and the brick is plotted in that column. Once the trend reverses, as each new brick is added on the downside, a new column is started. No column will every contain more than one brick. The result is a chart similar to the example above, where time scales are irregular and compressed. The net result is a strong filtering of the OHLC data, eliminating the ever present noise present in market data.

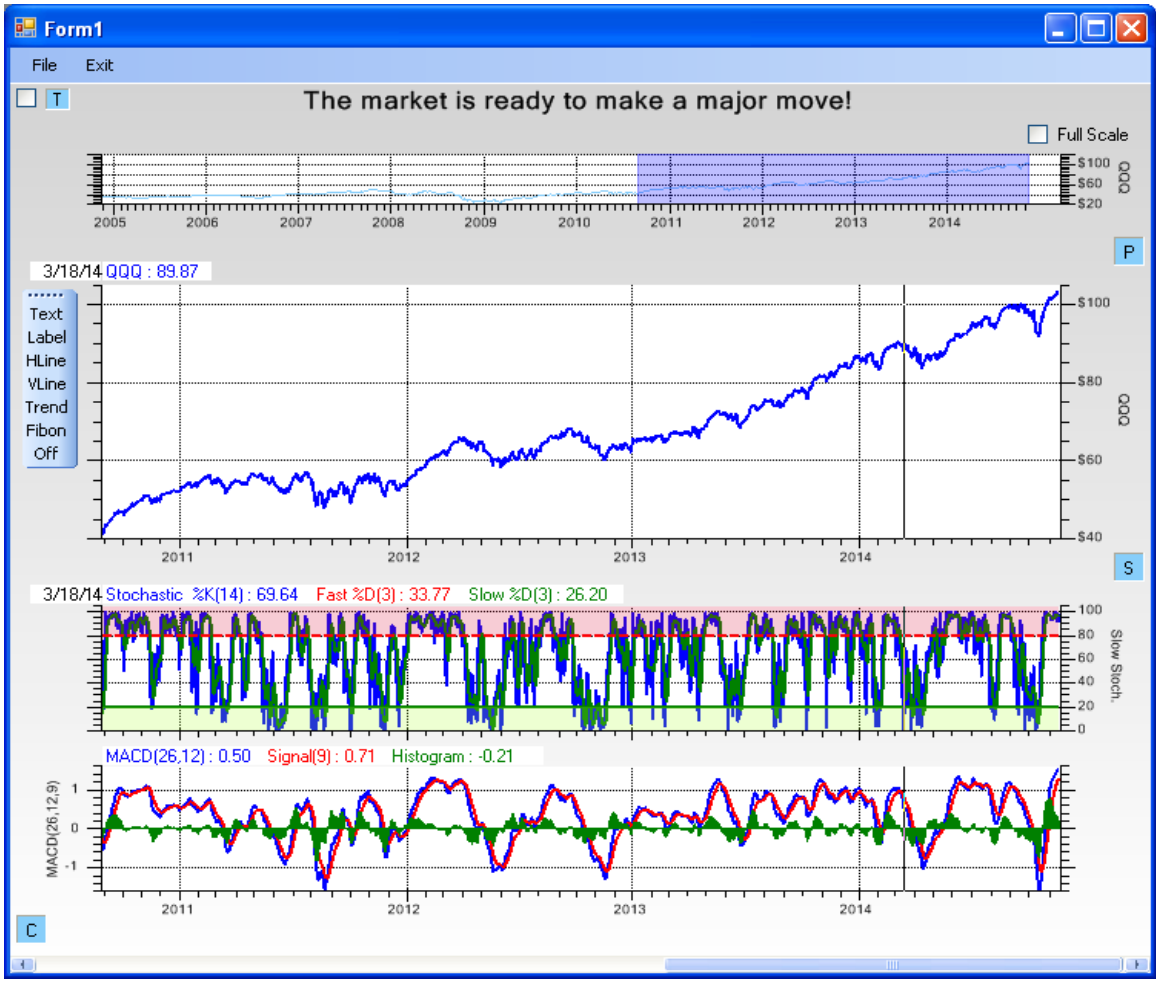
More detailed description of our Renko chart implementation is found in Chapter 10, *Renko Charts*.

Panning and zooming of data



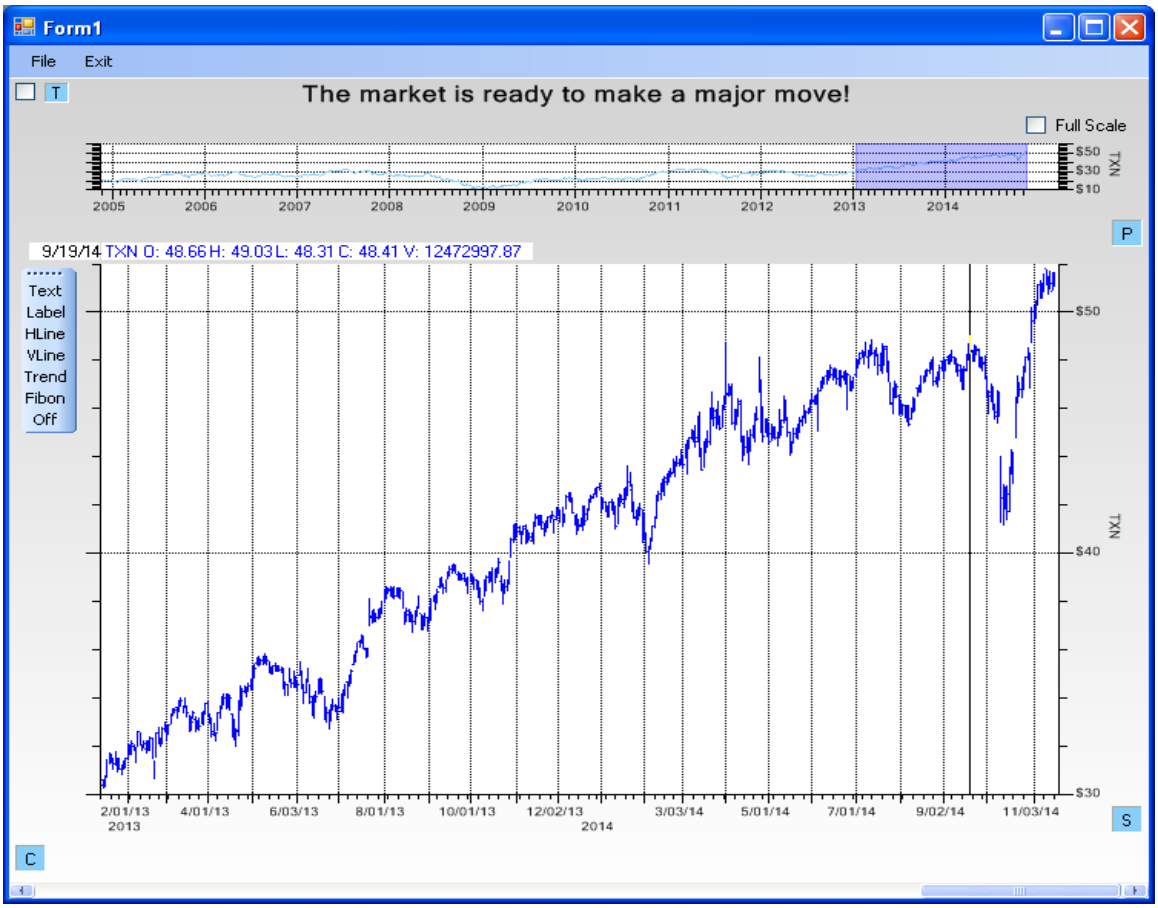
The top-most plot is called the Zoom window. It always displays all of the historical data for the primary ticker item. The user can expand, contract and move the zoom box (the filled transparent blue box) to adjust the view of the data.

Panning and zooming of OHLC data is can be accomplished several different ways. Above the primary chart you will see a short zoom chart which displays the full range of the OHLC data. The blue box in that chart represents the time frame of the data shown in the primary chart. First, you can click and drag on the zoom chart, creating a rectangle defining a custom time range, and the primary chart will update to reflect that time frame. Second, you can click and drag on the center of the zoom chart rectangle, and drag the entire rectangle to a new time start and end value, without changing the extent. And, you can click and drag on the start edge, or the stop edge, of the zoom rectangle, and expand or contract the zoom rectangle. Third, you can just use the scrollbar at the bottom of the FinChartView window to scroll through the data using a fixed extent for the time frame.



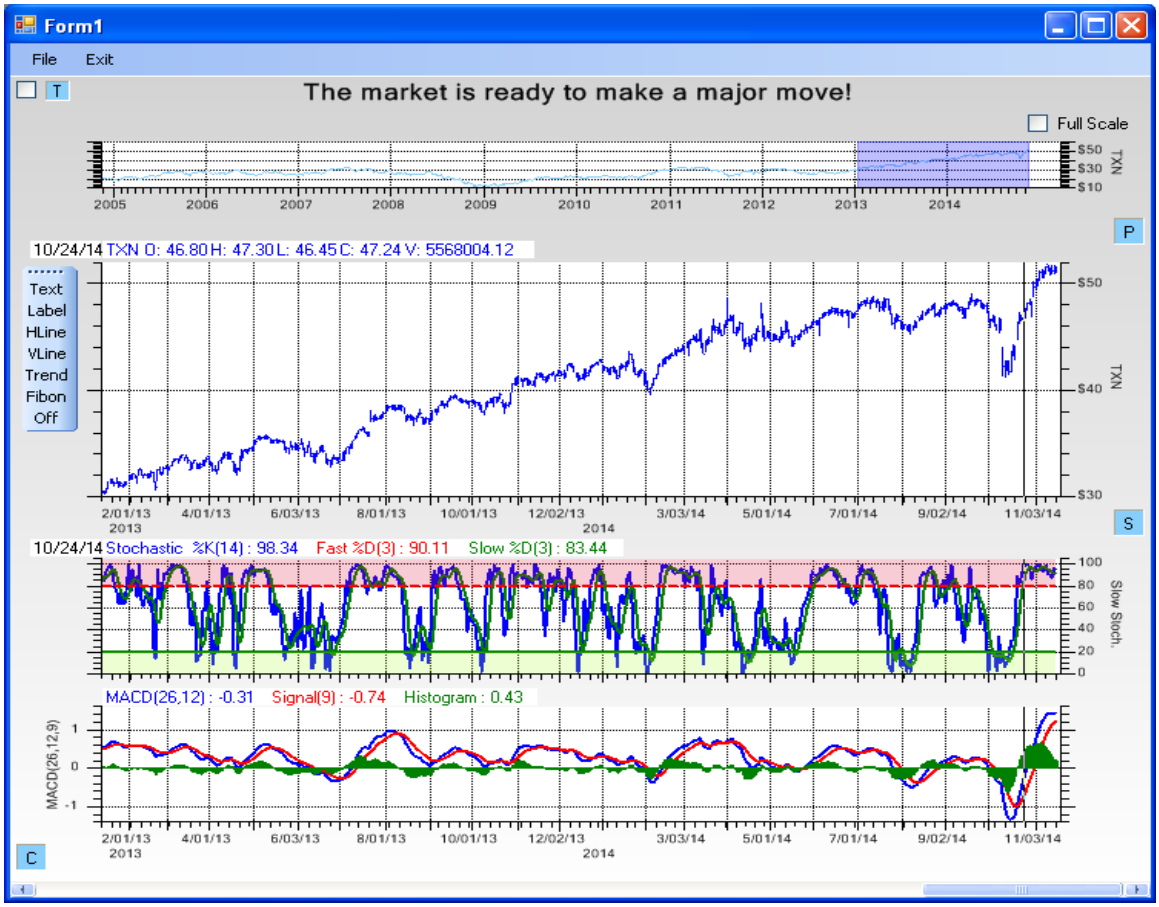
The zoom box of the of the Zoom window now encompasses almost four years of data. Changes to the zoom box affects both the Primary, and Secondary charts.

Data Cursor



Data values (Open-high-low-close-volume) for the time period under the cursor are displayed at the top of chart.

The data cursor is a mouse movable vertical line, which tracks the mouse cursor x location, drawn through all of the chart. Data values associated with the cursor location, appear to the left and above the plot area for each chart. This applies to the Primary chart, and the Secondary indicator charts. The data cursor values will be color coded to one of the plots in the associated chart.

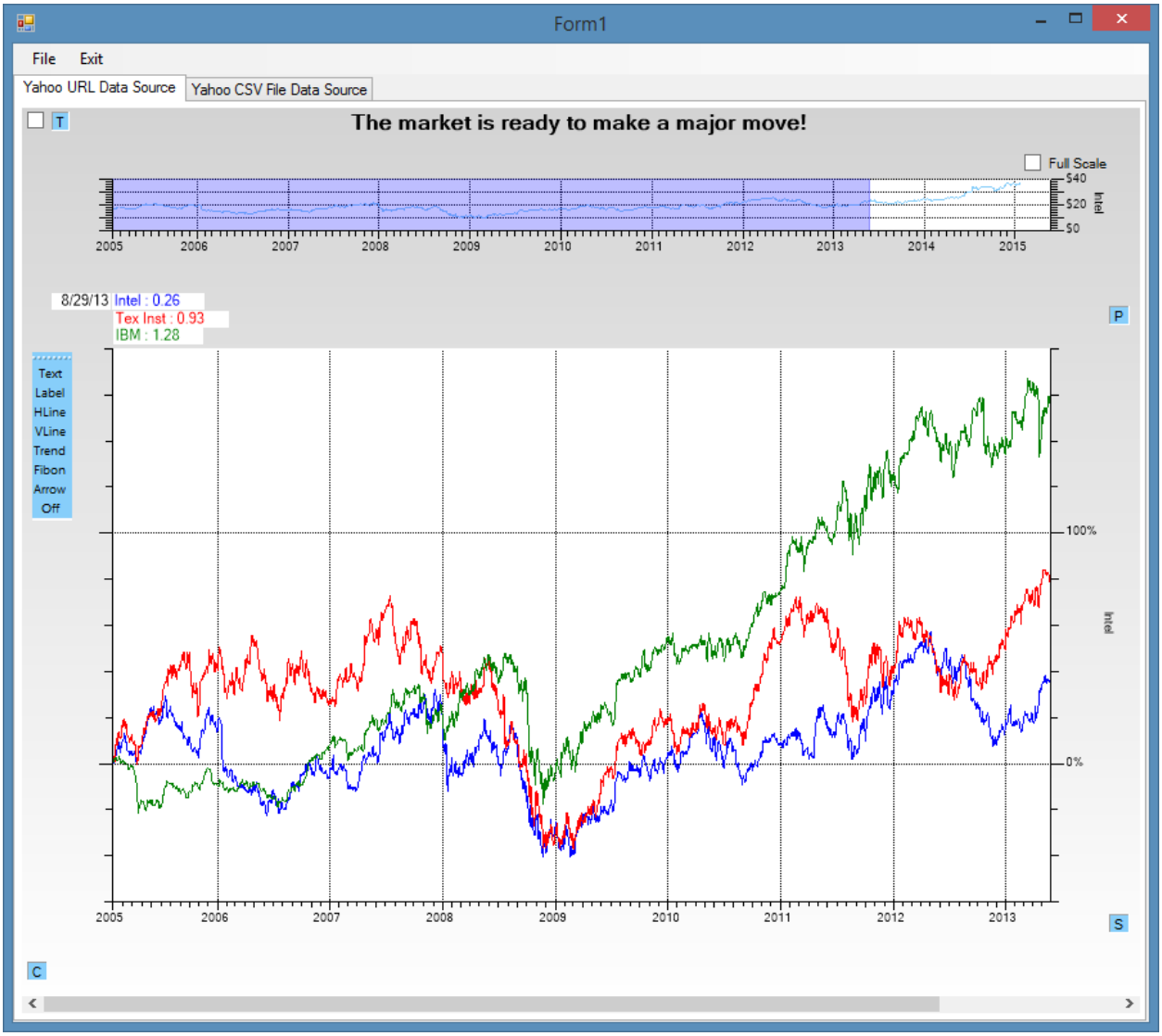


Data values for the plots are displayed for the Primary and Secondary charts.

Compare different stocks in the Primary Window

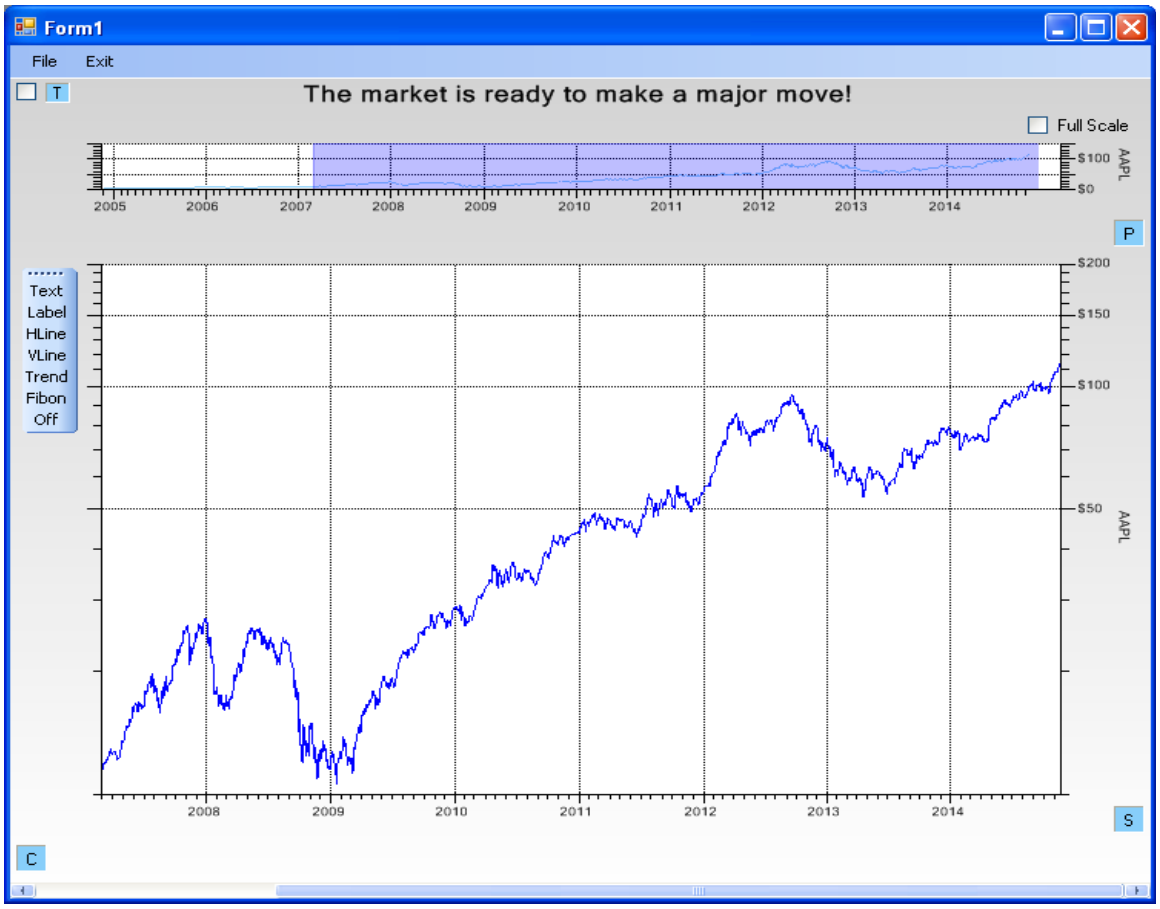
Comparing stocks with similar ranges using a simple linear scale.

Using the built-in setup dialog for the Primary Chart, you can select up to three different stocks for comparison in the Primary Chart window. This works OK if the stocks being compared are of the same range. If they are different ranges though, as in the case of plotting Apple (AAPL) or IBM (IBM) against other high tech stocks such as Intel (INTC) or Texas Instruments (TXN), the high price of IBM will swamp the detail of the lower prices stocks. In that case you should use a Normalized y-scale, which normalizes all of the stock gains against their initial values at the start of the data. That is what the chart below uses in comparing IBM vs TXN vs INTC.



Comparing stocks with different ranges using a normalized scale.

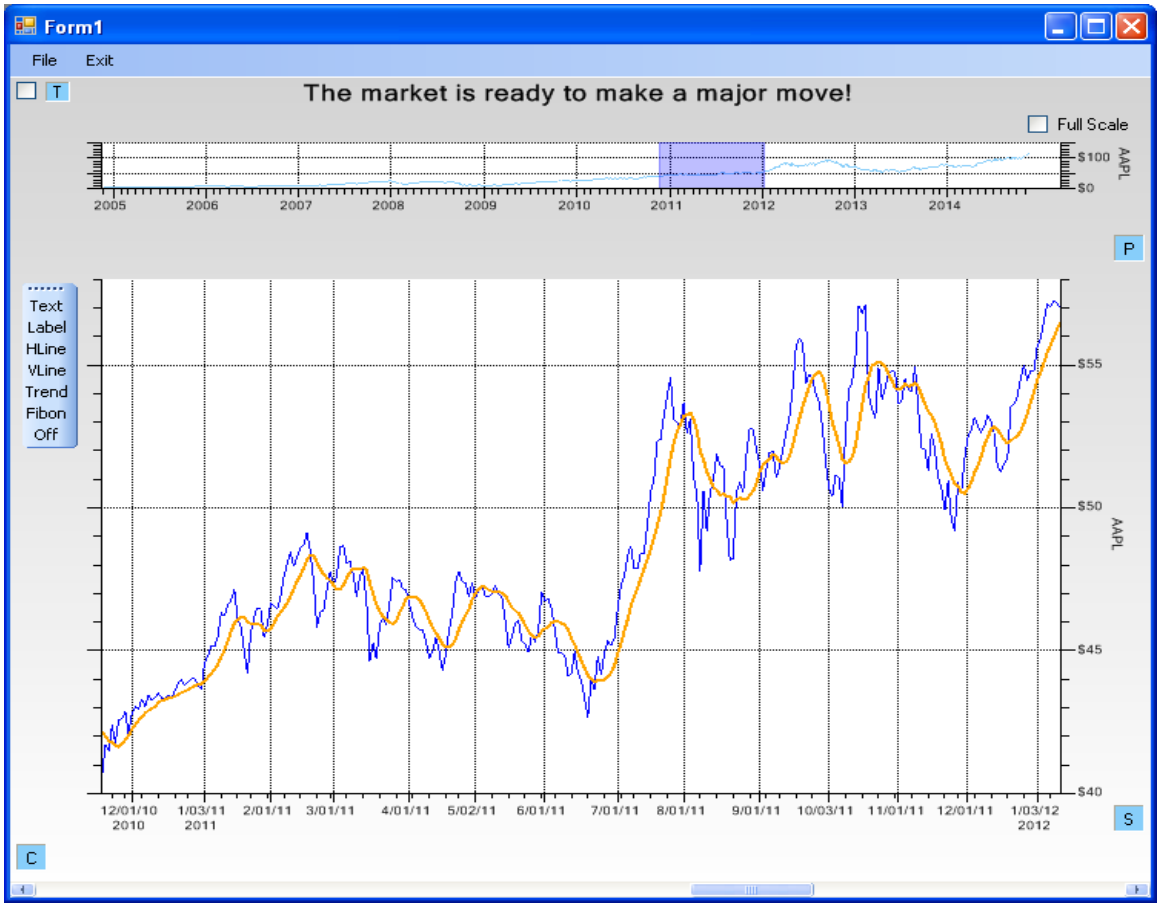
Y-Axis Scaling Options for the Primary Chart



A stock, such as Apple (AAPL), with a large range, is best displayed using a logarithmic y-axis scale.

There are three y-axis scaling options: Linear, Logarithmic, and Normalized. The default is the linear y-scale, which is the default for all Primary charts, and Indicator charts. The linear scale is probably the best one when you are looking at a single stock. If you are comparing multiple stocks, you will want to use the Normalized scale. The Logarithmic scale is good for looking at a stock such as Apple (AAPL) which has seen exceptional gains over the time frame under consideration. You can still see detail in both the low end and high end of the chart. See the chart above for an example of Logarithmic scale.

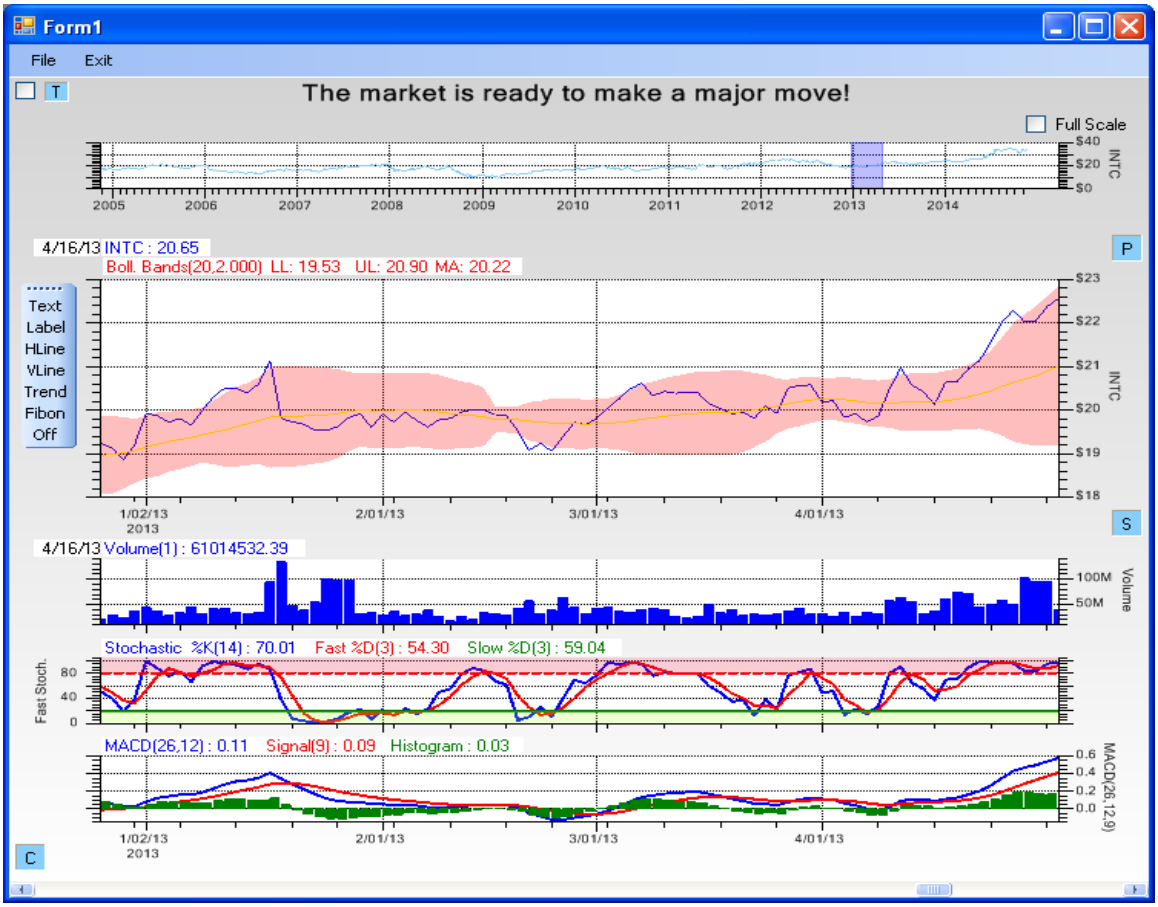
Dynamic auto-scaling to displayed data



The y-axis scale will auto-scale to the displayed data, no matter where you scroll to, or how many traces are displayed.

All charts will dynamically auto-scale to the displayed range of data. So even if the stock OHLC source has prices in the range 10 to 700, if the current three month view has a range of 500-700, the chart will automatically scale for 500 to 700, displaying the data with maximum resolution.

Synchronize scrolling (panning) and zooming of all chart windows



The scrollbar at the bottom and the Zoom window at the top control the current view of the data.

The Zoom window at the top will shift the time scale left or right, just like the scrollbar, if you click and drag the zoom box. If you click and drag on the left or right edge of the zoom box it changes the extent of the time scale.

The Primary chart window, and the Secondary indicator charts are synchronized. Only the zoom window above the Primary chart is fixed. If you pan left or right using the scroll bar, or the zoom chart, or change the chart range using the zoom chart, all of the charts will show the same time frame, with data values aligned vertically from chart to chart. A vertical data cursor will display data values for the chart at the top of each charts plot area.

Technical Indicator overlays for the Primary chart

Simple moving averages

Simple moving averages (SMA) are a easy way to filter out random noise, or price fluctuations, from a signal. Filtering makes it easier to identify short and long term trends in stock movement. A single simple moving average (MA) is often compared to original signal. When the original signal passes up and through the SMA, that can be considered a buy signal.

It is often considered a buy signal when the stock prices breaks through the 50 day moving average.

It is considered a buy signal when the blue OHLC data passes up and through the 50 point SMA above. The opposite is also true; it is considered a sell signal if the signal drops below the SMA.

Often times two or more moving averages of different lengths, acting on the same underlying signal, are compared and conclusions made based on intersection points between the SMA signals. If the shorter of two SMA signals passes up and through the longer, it is considered a buy signal. If the shorter of the two SMA signals passed down and through the longer, it is considered a sell signal.

It is often considered a buy signal if a short term simple moving average passes up and through a long term moving average.

In the picture above, the red line is a 15 point moving average, and the green line is a 50 point moving average. When the 15 point moving average pass up and through the 50 point moving average, that is considered a buy signal.

A SMA signal is simple to calculate. It is the unweighted mean of the previous n data items. The N period SMA for a time series Y(t) it can be calculated using the formulas

Starting with i = N (the SMA signal starts at the Nth data point)

$$i = N$$

$$SMA[i] = (Y[i] + Y[i-1] + Y[i-2] \ldots Y[i-N+1]) / N$$

When calculating successive values, a new value comes into the sum and an old value drops out, meaning a full summation each time is unnecessary for this simple case,

$$i > N$$

$$SMA[i] = SMA[i-1] - Y[i-N] / N + Y[i] / N$$

Exponential moving averages

The disadvantage of simple moving averages (SMA) is that it gives equal weight to all event items. If you are using a 200 period SMA, then the oldest data point 200 days ago is given the same weight in the calculation as the most

recent. Most practitioners of technical analysis are going to want more recent data values to have a greater influence on the indicator than older values.

The exponential moving average (EMA) mathematically weights more recent data more than older data. Because of this it can react quicker to changing circumstances. Otherwise, the buy and sell rules are much the same as the SMA. If the shorter of the EMA signals moves up and through the longer, it considered a buy signal. If the shorter of the EMA signals moves down and through the longer, it considered a sell signal. It can be used singly or in pairs, same as the SMA examples.

May traders prefer the exponential moving average (EMA) over the simple moving average (SMA), because the EMA weighs more recent data more heavily than old data.

A EMA signal is simple to calculate. The EMA for a time series Y(t) it can be calculated using the formula:

Starting with i = N (the EMA signal starts at the Nth data point), the first value is calculated as the simple moving average (SMA) of the same time period. This gives a starting point for the EMA calculation.

$$i = N$$

$$EMA[i] = SMA[i]$$

Subsequent calculations use the formula.

$$i > N$$

$$EMA[i] = EMA[i-1] * (1 – alpha) + Y[i] * alpha$$

where EMA[i] is the current EMA value

EMA[i-1] is the previous EMA value

alpha is the smoothing constant in the range 0.0 to 1.0

Y[i] is the current value of the source signal

In words, the EMA calculation is the current value of the source signal (the stock closing price), multiplied by the *alpha* value, plus the previous value of the EMA calculation, multiplied by (1 – *alpha*). The net effect is a weighted moving average where the older the source signal value is, the less it contributes to the current EMA value.

For a given signal, the only unknown in the equation above is the smoothing constant *alpha*. Financial technicians use a simple formula to calculate this value. If you specify a N point EMA, this implies that the alpha value is equal to 2 / (N + 1). For example, a 50 period EMA is equal to 2 / (50 + 1) = 0.0392 . Specifying a 50 period EMA is the same as specifying a 0.0392 alpha value for the EMA calculation. You won't see the alpha value referenced anywhere else. We will stick with the standard way to define a financial, technical analysis, EMA, which is to say that it is a N-point EMA.

Moving Average Bands

Moving Average Bands (or Envelopes) are formed by calculating a SMA (usually a 20-period average) on a source signal, and then forming two bands above and below the SMA signal by adding and subtracting a percentage deviation (usually in the range 1% to 10%) from the SMA signal. Moving average bands serve as an indicator of overbought or oversold conditions, visual representations of price trend, and an indicator of price breakouts.

Traders look for when the stock price breaks out of the moving average bands.

Trading Strategies Moving Average Bands

When the stock price does not appear to be trending up or down:

> Buy when the Low of the OHLC stock price penetrates the lower envelope and closes back inside the envelope.
>
> Sell when the stock price High of the OHLC price penetrates the upper envelope and then closes back down inside the envelope.

When the stock price breaks out above or below the envelop – trending one direction or the other:

> Buy when the prices break above the upper envelope
>
> Sell when prices break below the lower envelope

Moving Average Band Formulas

Moving average bands are simple to calculate. Start with a SMA of the signal (usually the Close value of the OHLC) and a bandwidth of BW. The two bands are:

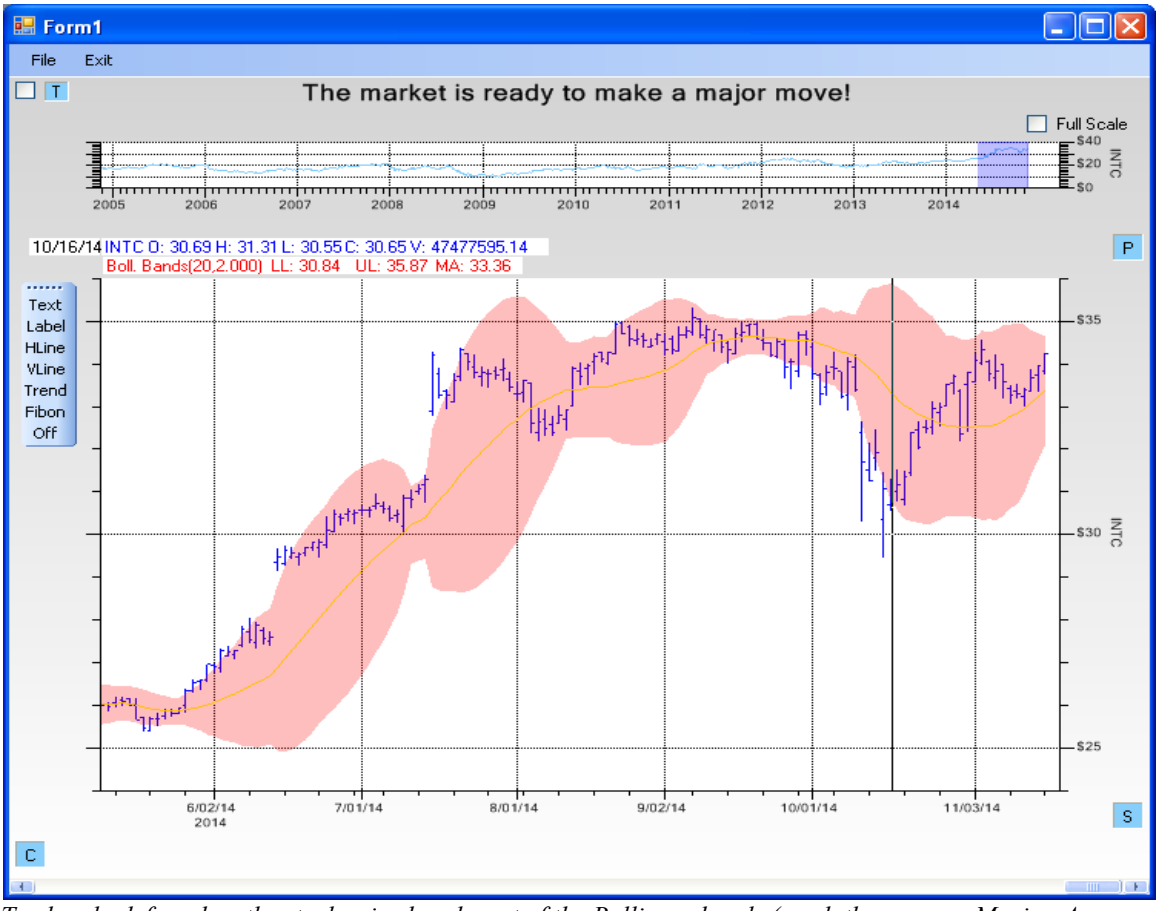
for all i

UpperBand[i] = SMA[i] * (1 + BW)

LowerBand[i] = SMA[i] * (1 - BW)

Bollinger Bands

Bollinger Bands (or Envelopes) are similar to Moving Average bands, except they add a little statistical science to the formation of the indicator. Bollinger Bands still use a SMA calculation for the central line (20 period SMA is standard). But instead of using a fixed percentage as the band width, it defines the separation between the two bands using a multiple of the standard deviation signal, calculated using the previous N periods of the closing value of the OHLC data. The theory is that stock price action follows a normal distribution about the mean, and 95% of a stocks price movement should fall within two standard deviations, plus and minus, of the mean value. Price action outside of the +-2 standard deviations band signifies that something unusual happening and that you expect some sort of break in the current trend.



Traders look for when the stock price breaks out of the Bollinger bands (much the same as Moving Average bands).

The filled area represents the Bollinger bands. Usually Bollinger bands include the SMA line, orange in this case.

Trading Strategies Bollinger

Interpretations of Bollinger Bands vary, but one is that when stock prices push up and through the upper band, the stocks are thought to be overbought. And when stock prices push down through the lower limit, stocks are thought to be oversold. Therefore:

Sell when the stock prices push upward through the upper limit.

Buy when the stock prices push downward through the lower limit.

There are entire books written about Bollinger Band trading strategies, including one by John Bollinger, "Bollinger on Bollinger Bands", the man who invented the indicator. So don't expect their practical use to be as cut and dry as the buy/sell signals described above.

Bollinger Band Formulas

K = Standard deviation multiple - usually equal to 2

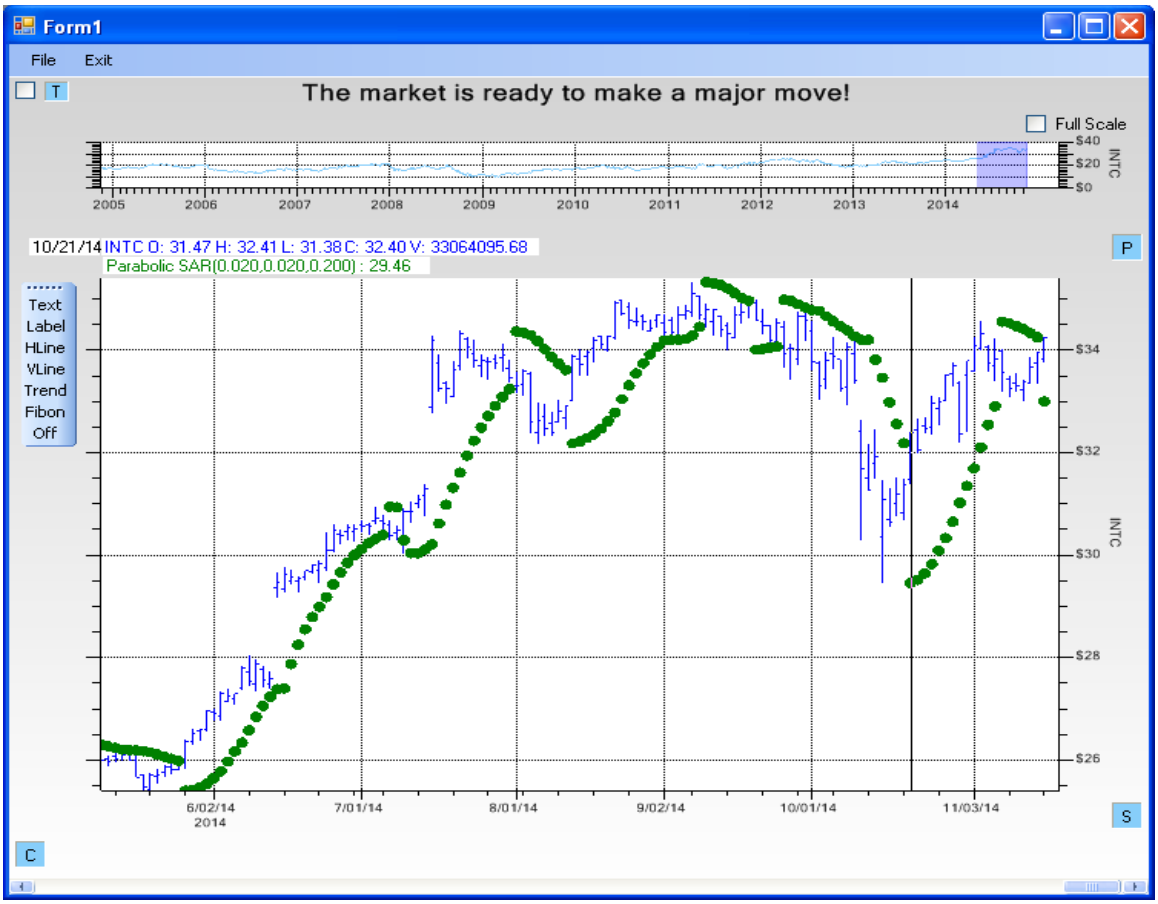
for all i

SD = StandardDeviation of the previous N (usually 20) values in the N period SMA

UpperBand[i] = SMA[i] * (1 + K * SD)

LowerBand[i] = SMA[i] * (1 – K * SD)

Parabolic SAR



The Parabolic SAR is used by traders to set stop loss orders for a stock.

In the example above, the green dots represent the Parabolic SAR indicator

The Parabolic SAR (Parabolic Stop and Reverse) is probably the strangest looking indicator. The Parabolic in the name comes from the accelerating rise and fall of the indicator around the source signal.

The well known market technician J. Welles Wilder created the indicator and described it in his book *New Concepts in Technical Trading Systems*. Published in 1978, the book also describes a number of other Welles indicators, including the Average True Range, the Directional Movement Index and the Relative Strength Index.

When the parabola is below the price action it is considered bullish, and when i it is above the price action, it is considered bearish. Since it forecasts one day in advance, traders often use the Parabolic SAR value to set stop limits for trades that day. For example, in an upward trend, if the stock price falls below the P SAR value for that day, sell the stock in order to protect gains.

Parabolic SAR Formulas

$PSAR_n$ (Parabolic Stop And Reverse) = today's value of the Parabolic SAR

$PSAR_{n+1}$ = Tomorrow's value for the PSAR. The PSAR formula forecasts one day in advance

EP (Extreme Point) = The highest high of the current uptrend or the lowest low of the current downtrend.

AF (Acceleration Factor) = Determines the sensitivity of the PSAR. AF starts at .02 and increases by .02 every time the EP rises in a Rising PSAR or EP falls in a Falling PSAR. The maximum value is usually clamped at 0.20. The starting value, 0.02, step value 0.02 and maximum value 0.20, are all variables in the software and you can choose anything you want.

$PSAR_{n+1} = PSAR_n + AF_n*(EP_n - PSAR_n)$

where

AF is calculated according to the rules described above..

Special conditions which override the calculated value

- If in an upward trend, the new PSAR value is calculated and if the result is more than today's or yesterday's lowest price, it must be set equal to the lower of those two days. Some texts (not Wilder's) say to set it to the closer of the two lows. We use the lower of the two lows in this software.

- In a downward trend the new PSAR value is calculated and if the result is less than today's or yesterday's highest high price, it must be set equal to the higher of those two days. Some texts (not Wilder's) say to set it to the closer of the two highwe. We use the higher of the two highs in this software.

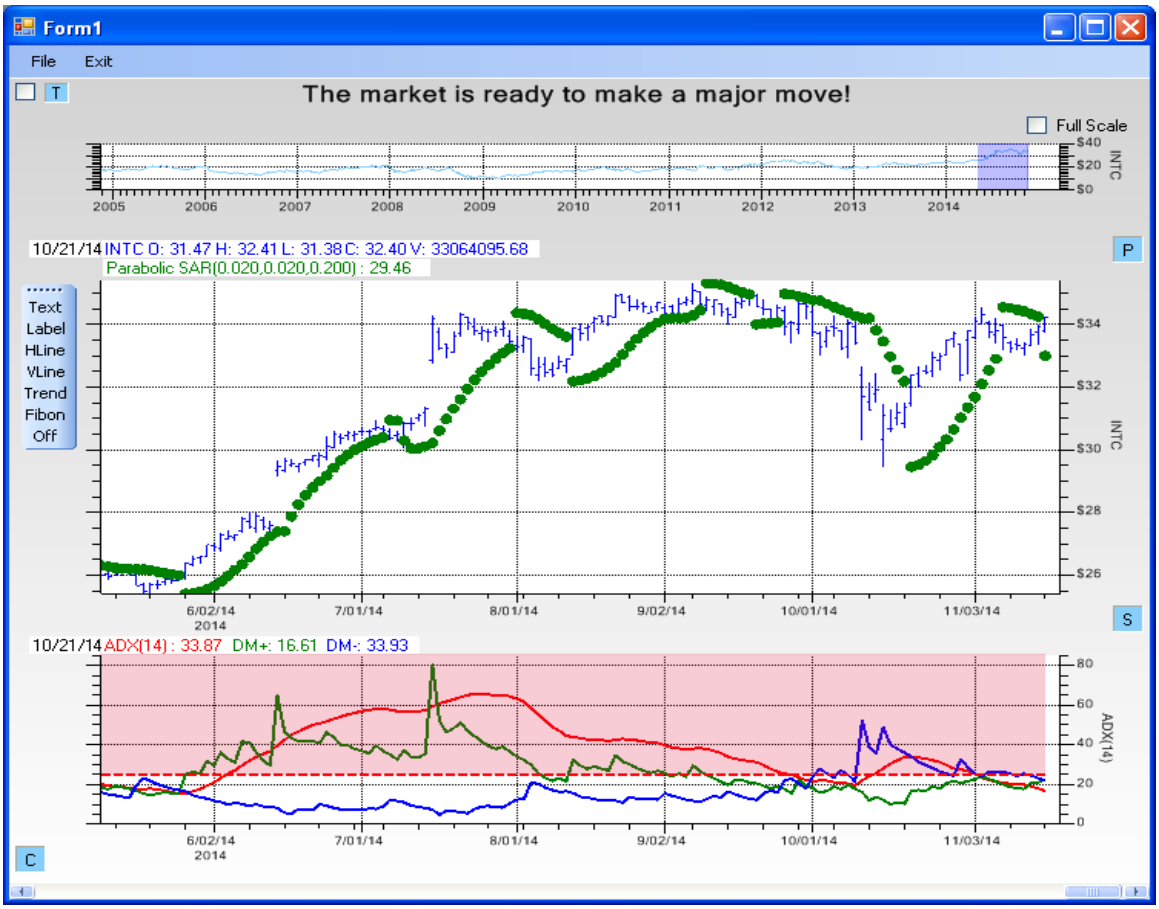
- If the next period's PSAR value is inside (or beyond) the next period's price range, a new trend direction is then signaled. The PSAR must then switch sides.

- Upon a trend switch, the first PSAR value for this new trend is set to the last EP recorded on the prior same direction trend, EP is then reset accordingly to this period's maximum, and the acceleration factor is reset to its initial value of 0.02. On re-calculation of the new SAR after the switch, some packages will check to see if the new EP changes (rises in an uptrend or falls in a down trend) first, and boost the acceleration factor on the first calculation, making it 0.04. We always force the first calculation after a switch-over to use an acceleration factor of 0.02, and on subsequent calculations within the trend we increase it a step at a time as the EP changes.

Secondary windows technical indicators

Underneath the primary chart you will find zero or more secondary charts. The secondary charts are used to display technical indicators which are best displayed in their own chart area, usually because they use a y-axis coordinate system which is not the same as the primary chart. The secondary charts always key on the first stock in the primary chart, so you cannot monitor one stock in the Primary chart (INTC for example) and another in the secondary charts (TXN for example). Chapter 7 describes programming the Secondary Chart windows in detail.

Average Directional Indicator (ADX)



The ADX indicator is used to measure the strength of a trend.

The Average Directional Indicator (ADX) was originally developed by J. Welles Wilder and is described in his book "New Concepts in Technical Trading System". The ADX is used to measure the strength of a trend. Its related components, the Minus Directional Indicator (-DI) and Plus Directional Indicator (+DI), are used to measure the strength of a trend. The ADX indicator is often used in conjunction with the Parabolic SAR indicator, to confirm a trend. In the chart above, the Primary chart shows the Parabolic SAR indicator, and the bottom chart shows the ADX indicator (blue line), the -DI indicator (green line) and the +DI indicator (red line);

A stock is said to be in a strong trend if the ADX value is greater than 25 (some use 20 as the threshold). Note, that the ADX indicator does NOT specify a trend direction. So a strong positive trend, or a strong negative trend, will result in ADX values greater than 25 (20). So the ADX value can signal traders whether or not they should be using trend trading strategy.

Assuming you are in an upward trend confirmed by an ADX value of 25 (or 20), a buy signal occurs when +DI crosses above – DI, with a stop loss place on the low of the day. The buy signal remains as long as this low holds, even if +DI crosses back below - DI. If the stock price falls below the stop loss low, the buy signal is ended. If the trade become profitable, traders should uses stop losses (perhaps the Parabolic SAR value) to protect their profit. A sell signal occurs when - DI crosses above +DI.

ADX, +DI and -DI Formulas

The ADX indicator is a slow moving signal, because it uses long exponential smoothing periods. Also, it uses

unique variant of exponential smoothing, called Wilder smoothing. As discussed in the Exponential Moving Average section, the standard exponential smoothing equation is:

EMA[i] = EMA[i-1] * (1 – alpha) + Y[i] * alpha

alpha is the smoothing constant in the range 0.0 to 1.0

for regular (non-Wilder) time-based exponential moving average calculations, the alpha value is calculated as:

alpha = 2 / (N + 1)

where N is the number of time periods specified in the smoothing (in a 20 point exponential smooth of a set of data, the alpha value is calculated as (2 / 21) or 0.0952. But in the case of Wilder smoothing, the formal for alpha is

alpha = 1 / N

so that if the period length is 20, the alpha value of the exponential smoothing equations becomes 0.05.

The ADX indicator also uses something called the True Range, which is a value for each period calculated from the OHLC for that period and the previous period. The formula is:

for i>0

True Range[i] = **Max**(Highs[i] – Lows[i], **Abs**(Highs[i] – Closes[i-1]), **Abs**(Closes[i-1] – Lows[i]))

Because the + and – signs in front of +DM and -DM can be confused with arithmetic operators, we rename those indicators pDM and mDM for remaining formulas.

for i>0

pDM[i] = Highs[i] - Highs[i-1] Positive in a rising trend
mDM[i] = Lows[i-1] - Lows[i] Positive in a falling trend

An inside day (where both calculations are negative) both values are set to 0.

If pDM[i] >= mDM, then pDM[i] remains at its calculated value, mDM[i] is set to 0.0.

If pDM[i] < mDM, then mDM[i] remains at its calculated value, mDM[i] is set to 0.0.

Calculate the True Range values for the OHLC data

TR14 = 14 period Wilder EMA of True Range data

Calculate the 14 Period Wilder EMA of the pDM and mDM data

pDM14 = 14 period Wilder EMA of pDM data

mDM14 = 14 period Wilder EMA of mDM data

Calculate the +DI and -DI indicators

for i > 0

pDI14[i] = pDM14[i] divided by TR14[i]

mDI14[i] = mDM14[i] divided by TR14[i]

Then, calculate the components of the Average Directional Movement Index (ADX):

ADX[i] = 100 * ABS((pDI14[i] - mDI14[i]) / (pDI14[i] + mDI14[i]))

ADX14 = 14 period Wilder EMA of AD data

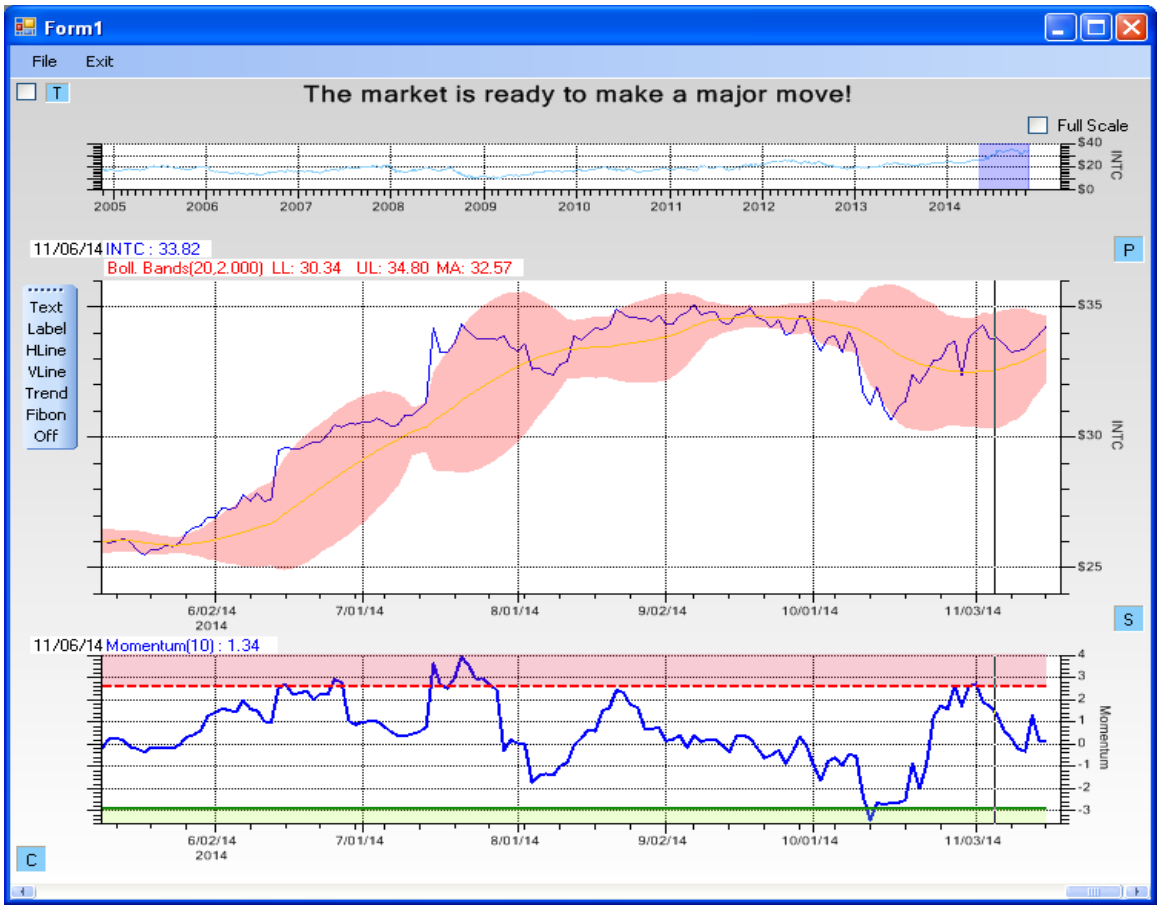
The actual returned values are the pDI14 array, the mDI14 array and the ADX14 array, all representing 14 period Wilder EMA of the underlying index.

Special Note

The proper use of the ADX indicator always requires that your historical data start at least 150 periods (days) prior to the time period you are interested in. Because it takes that long for the exponential smoothing averages to settle down. Results where your data starts only 50 days before the the time period you are interested will be similar to, but not the same as the results where a 150 day lead time is used.

Momentum (also known as the Change Indicator)

The Momentum indicator is used to identify the speed (or strength) of a price movement. It is calculated as the difference between today's closing price and the close N days ago.



While the Momentum indicator is positive, it designates an continuing upward trend in prices. A negative upward trend designates a continuing downward trend.

Trading Strategies Momentum

Action Limits
Set high (overbought) and low (oversold) thresholds for action. No firm algorithm is available for this. Some suggest setting the thresholds at 67% the high and low peak values on both sides of 0.0.

Buy when
The Momentum indicator drops below the lower threshold and then rises back above it

Sell when
Momentum crosses to above the overbought level and then falls back below it.

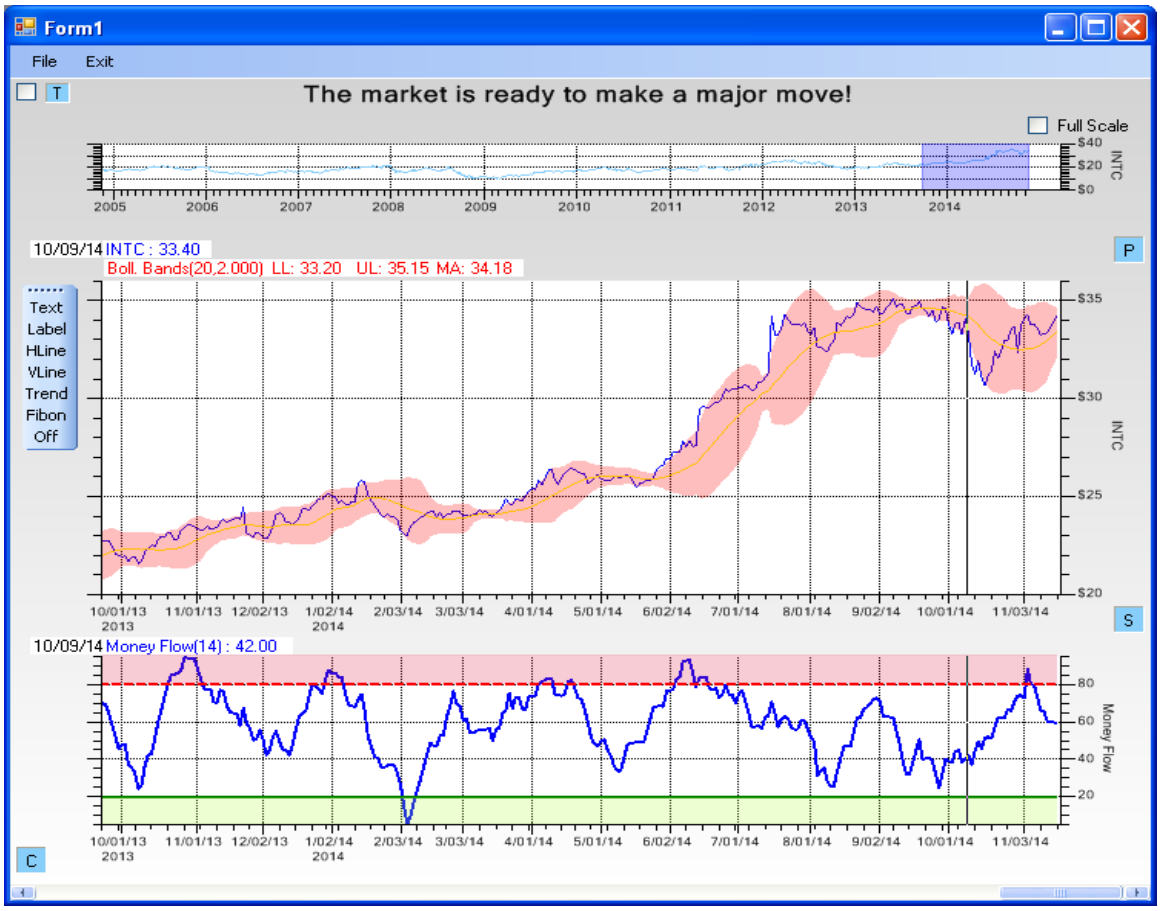
On a bearish divergence - with the first peak above the overbought level.

Formula

For i >= N

Momentum[i] = (Close [i] - Close [i-N])

Money Flow (MFI)



The Money Flow indicator provides more information than the Relative Strength indicator, because it also takes into account volume.

The Money Flow indicator is considered an oscillator, cycling between 0 and 100. it was create by market technicians Gene Quong and Avrum Soudack. It uses both the Typical Prices for a period, and the period volume, in its calculation. It is sometimes referred to as a volume adjusted RSI indicator.

Trading Strategies Money Flow

Action Limits
Set high (overbought) and low (oversold) thresholds for action limits. Recommended value for the high limit is 80, and the low limit is 20.

Buy when
The Money Flow indicator drops below the lower threshold and then rises back above it.

A reverse of the indicator to the upside while the overall stock trend is down. The money flow reversal is considered a leading indicator of the future stock trend.

Sell when
The Money Flow indicator crosses to above the overbought level and then falls back below it.

A reverse of the indicator to the downside while the overall stock trend is up. The money flow reversal is considered a leading indicator of the future stock trend.

Formula

Calculate the Typical Price for each period = (High + Low + Close)/3

Calculate the Raw Money Flow for each period = Typical Price x Volume

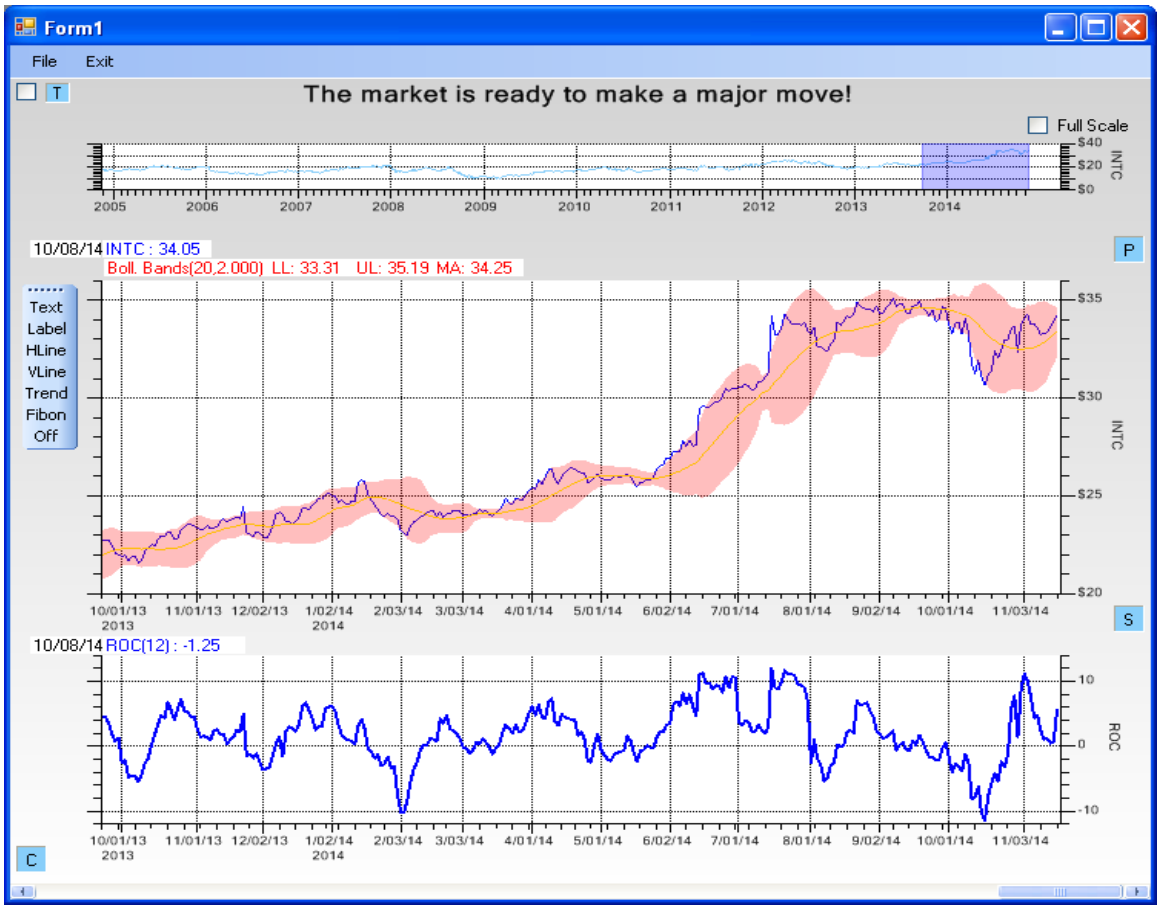
For each period, calculate as separate values the positive money flow over the previous N periods (usually 14), and the negative money flow over the same time frame.

Calculate the Money Flow Ratio as the ratio (Positive Money Flow)/(Negative Money Flow)

Normalize Money flow ratio for the range 0-100 using:

Money Flow Index = 100 - 100/(1 + Money Flow Ratio)

Rate of Change (ROC)



The Rate of Change indicator is a normalized version of the Momentum indicator.

The Rate of Change indicator is essentially a normalized version of the Momentum indicator. It takes the Momentum value at each time period and divides it by the Close price N days ago, where N is the same value used in the Momentum calculation. The result is multiplied by 100 to convert from a fraction to a percentage.

Trading Strategies Rate of Change

Buy when

When in an up trend, buy when the ROC value passes upward through zero.

Sell when

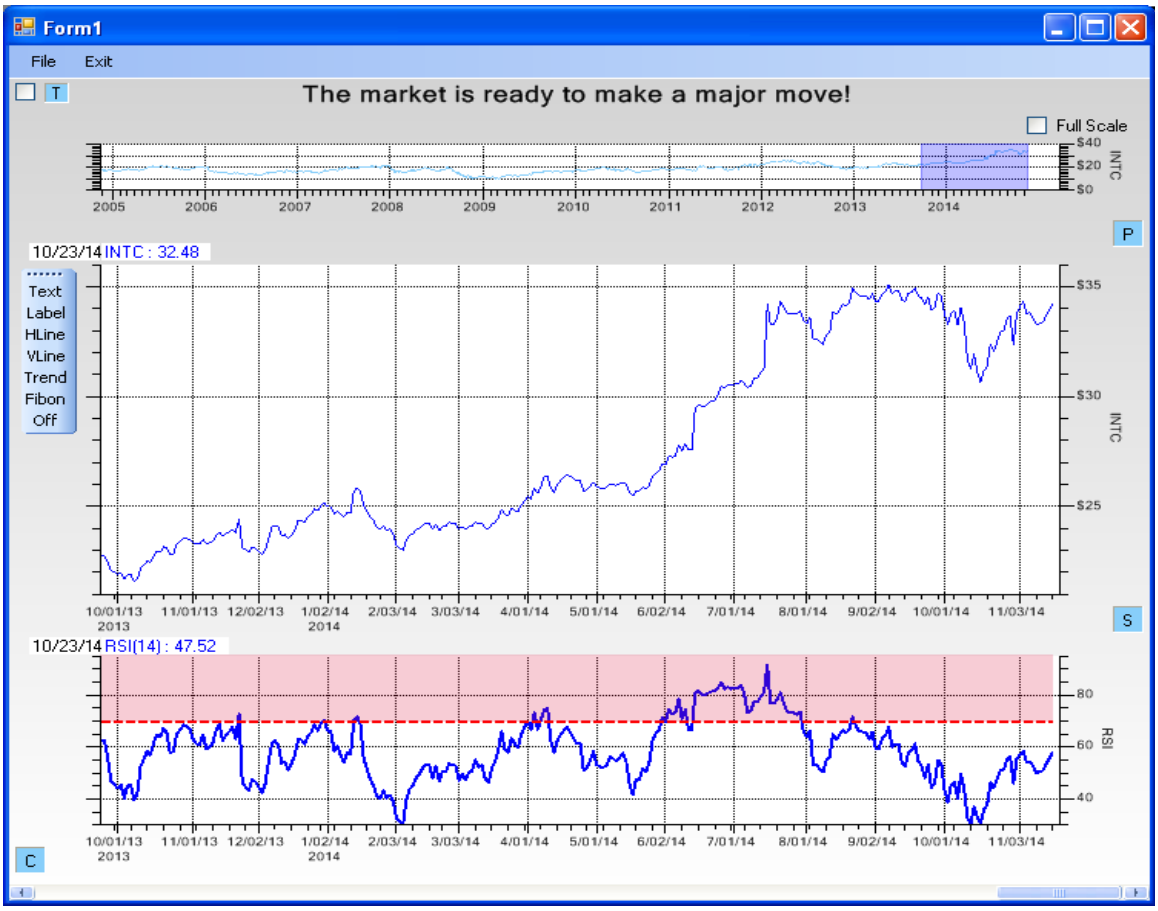
When in a down trend sell when the ROC value passes downward through zero

Formula

For i >= N

ROC[i] = 100 * ((Close [i] - Close [i-N])/ Close [i-N])

Relative Strength (RSI)



The Relative String Indicator compares the magnitude of gains (up-Closes) versus losses (down-Closes) for the time period under consideration, in order to determine when a stock is overbought or oversold.

The Relative Strength Indicator (RSI) was originally developed by J. Welles Wilder and is described in his book "New Concepts in Technical Trading System". It indicates strength or weakness in a security based on the closing price action within the specified trading period. The RSI is calculated using the ratio of higher closes to lower closes: stocks which have had more or stronger positive changes have a higher RSI than stocks which have had more or stronger negative changes. The RSI indicator is normalized so all values are percentages in the range 0 to 100. Intermediate values in the calculation are smoothed using the Wilder exponential smoothing method, usually using a 14 day period.

Trading Strategies RSI

Action Limits
Recommended action limits for RSI are a high limit of 70, and a low limit of 30.

Trading Strategies Rate of Change

In a non-trending market

Buy when

Buy when the RSI value passes upwards and through the low action limit (30).

Sell when

Sell when the RSI value passes downward and through the high action limit (70).

In a trending market

Buy when

Buy when the RSI value passes upwards and through the low action limit (40).

Sell when

Sell when the RSI value passes downward and through the high action limit (60).

Formula

For i >= 1

if (Close [i] - Close [i-1]) > 0

U[i] = (Close [i] - Close [i-1])

D[i] = 0;

else if (Close [i-1] - Close [i]) > 0

U[i] = 0

D[i] = (Close [i-1] - Close [i])

else if (Close [i-1] - Close [i]) == 0

U[i] = 0

D[i] = 0

Each U[i] and D[i] value is smoothed using an N period exponential smoothing routine. The result is

U[i]Sm

D[i]Sm

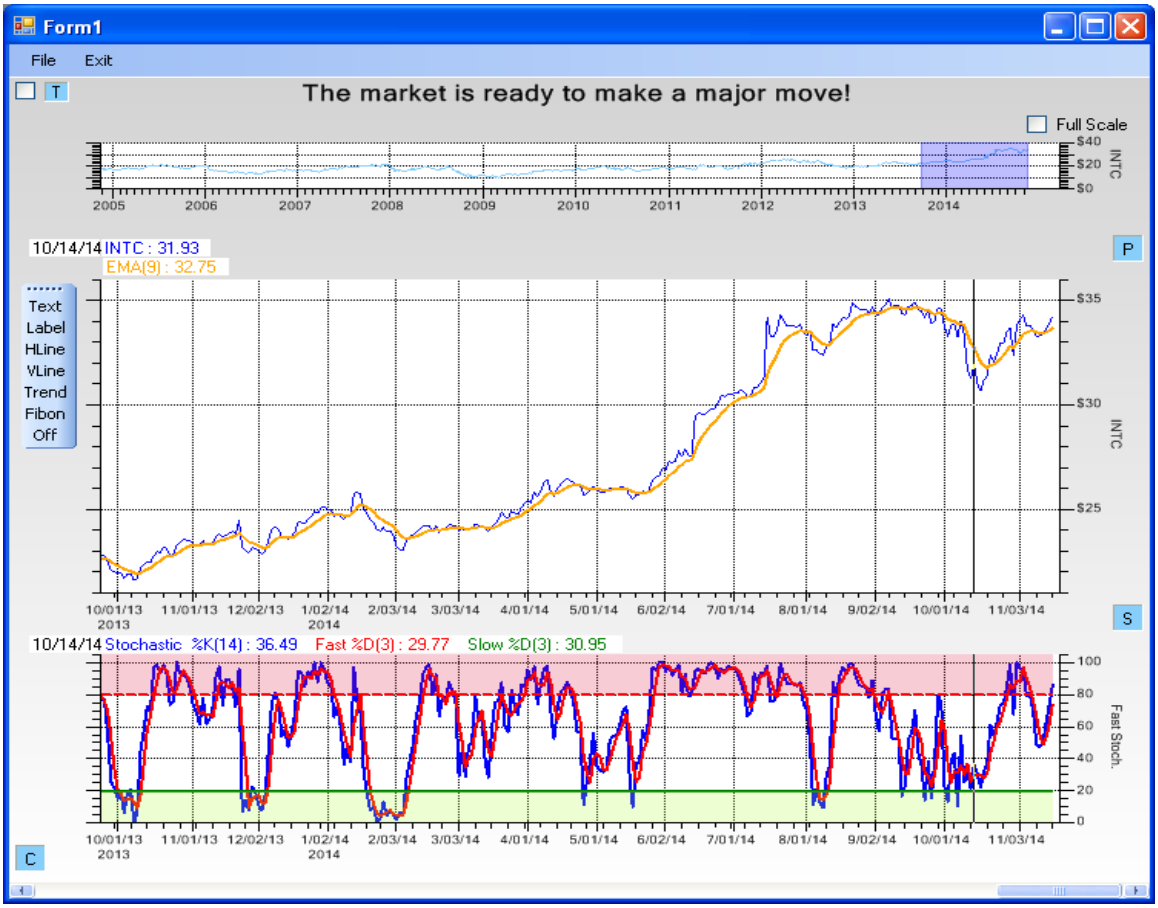
The RS value is calculated as

RS[i] = U[i]Sm / D[i]Sm

and finally, the RSI values is normalized to the range 0 to 100 using the formula

RSI[i] = 100 – (100 / (1 + RS[i]))

Stochastic (Fast and Slow)



The bottom chart plots %K vs %D Fast Stochastic

The Stochastic indicator was developed by George Lane, president of Investment Educators Inc, Watseka, IL. It is based on the assumption that as prices trend upwards, closing prices tend to be in the upper part of the periods OHLC price range. And the opposite is true, in down trends, closing prices tend to be in the lower part of the periods OHLC price range.

Formulas

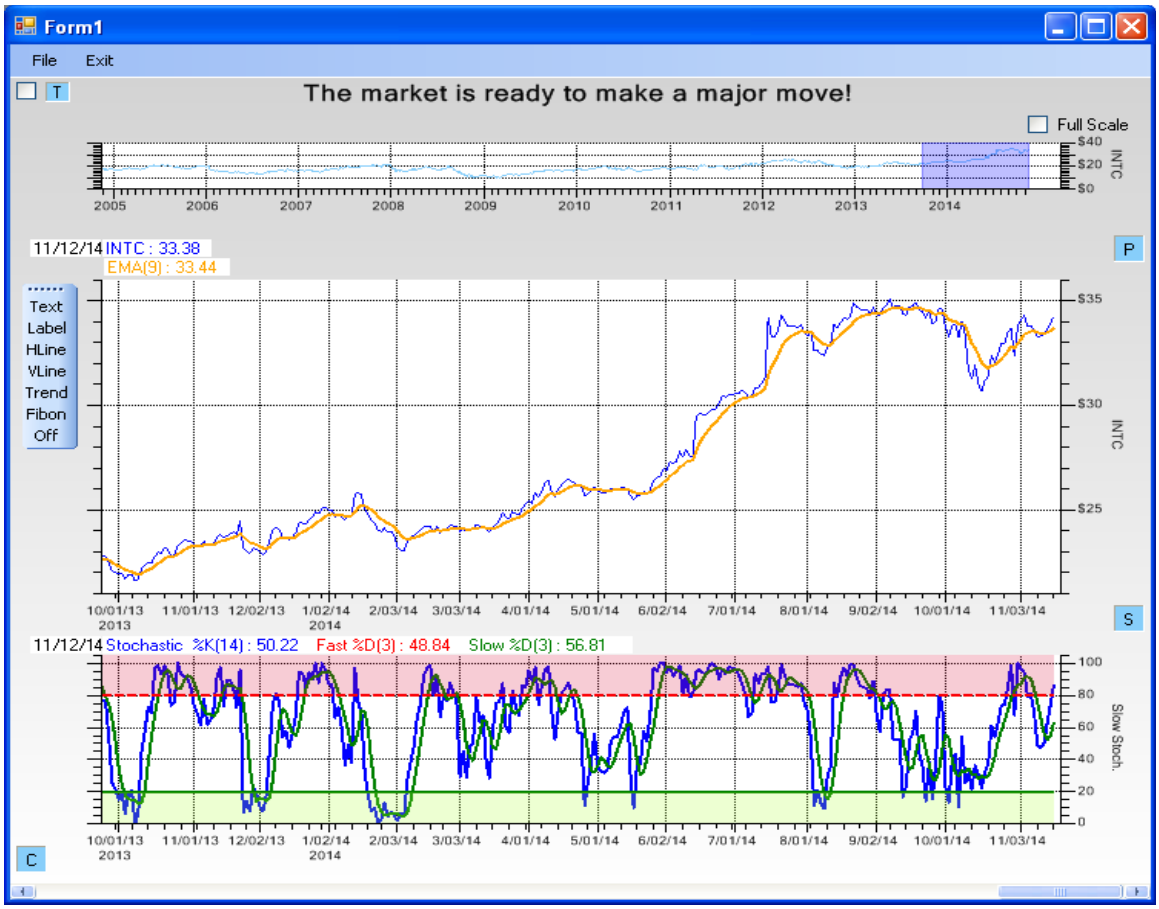
The first line in the Stochastic indicator is the %K line. The %K line is calculated as

$$\%K[i] = 100 * ((Close[i] - L14[i])/(H14[i] - L14[i]))$$

where Close[i] is the periods closing price, L14[i] is the lowest low for the last Nk periods, and H14[i] is the highest high for the last Nk periods, where Nk is usually 14 periods. This results in a value of 0 to 100. A %K[i] value of over 80 means that the closing price is very near the top of the OHLC range, and a value of under 20 means that the closing price is near the bottom of the OHLC range.

The second line in the Stochastic indicator is the %D line. The %D line is calculated as a simple Nd point moving average of the %K line, where Nd is usually 3 periods. So it is just a slower, smoother version of %K line. The 3 period smoothed version of %D is known as the *Fast Stochastic* indicator. If %D is smoothed using a 3 period

average, then it becomes the *Slow Stochastic* indicator. The Slow Stochastic indicator is the more widely used. One confusing item is that both the Fast Stochastic and the Slow Stochastic are referred to as %D.



The bottom chart plots %K vs %D Slow Stochastic. If you compare it to the previous chart you will see that the red line is more smoothed, and lags a bit.

Trading Strategies Stochastic

Action Limits
Recommended action limits for Stochastic charts are a high limit of 80, and a low limit of 20.

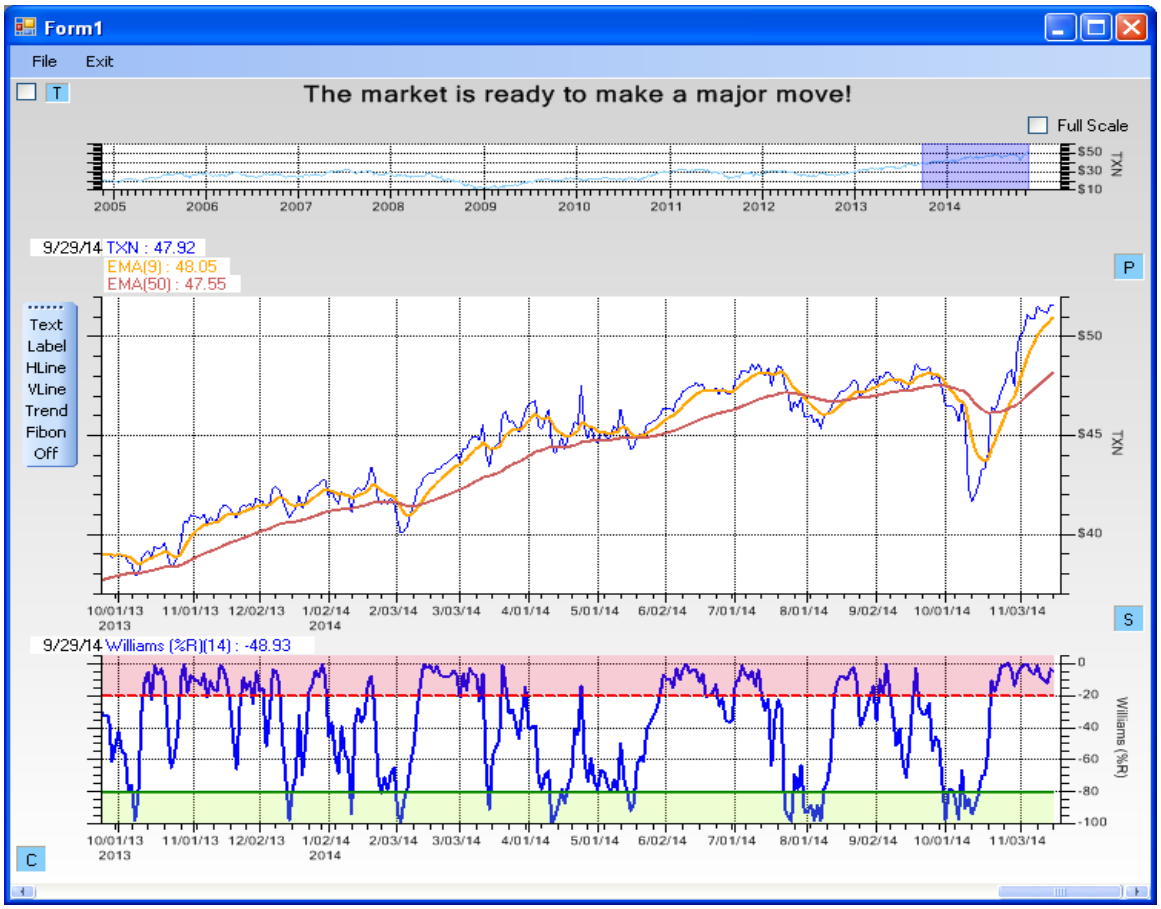
Buy when

Buy when the %K line passes up and through the %D line when the %D line is below the low (20) action limit value.

Sell when

Sell when the %K line passes down and through the %D line when the %D line is above the high (80) action limit value.

Williams %R



The Williams %R indicator is used to detect overbought and oversold conditions for the given stock.

Williams %R was developed by Larry Williams, a publisher of trading and technical analysis materials, to indicate overbought and oversold market conditions for a stock. Similar to the Stochastic indicator, it compares a stock's close to the high-low range over a certain period of time, usually 14 days. For some strange reason the indicator is normalized in the range 0.0 to -100, so that an overbought condition is represented by the indicator above the -20 line, and an oversold condition represented by the indicator below -80. A value of -100 for the index means that the close today was at the lowest low of the past N days, and a value of 0 for the index means that a close today was at the highest high of the past N days. When it is plotted the Williams %R indicator looks the same as the Stochastic %K indicator, except that the y-axis values are shifted down by 100.

Trading Strategies Stochastic

Action Limits

Recommended action limits for Williams %R charts, are a high limit of -20 (overbought), and a low limit of -80 (oversold). While Williams himself specifies a 10 day period, most users prefer the 14 day period, the same as the Stochastic %K indicator.

Williams used a 10 trading day period and considered values below -80 as oversold and above -20 as overbought. Most users seem to use a 14 day period though, consistent with the Stochastic indicator time period.

Buy when

Buy when the %R line passes down and through the low (-80) action limit value.

Sell when

Sell when the %R line passes up and through the high (-20) action limit value.

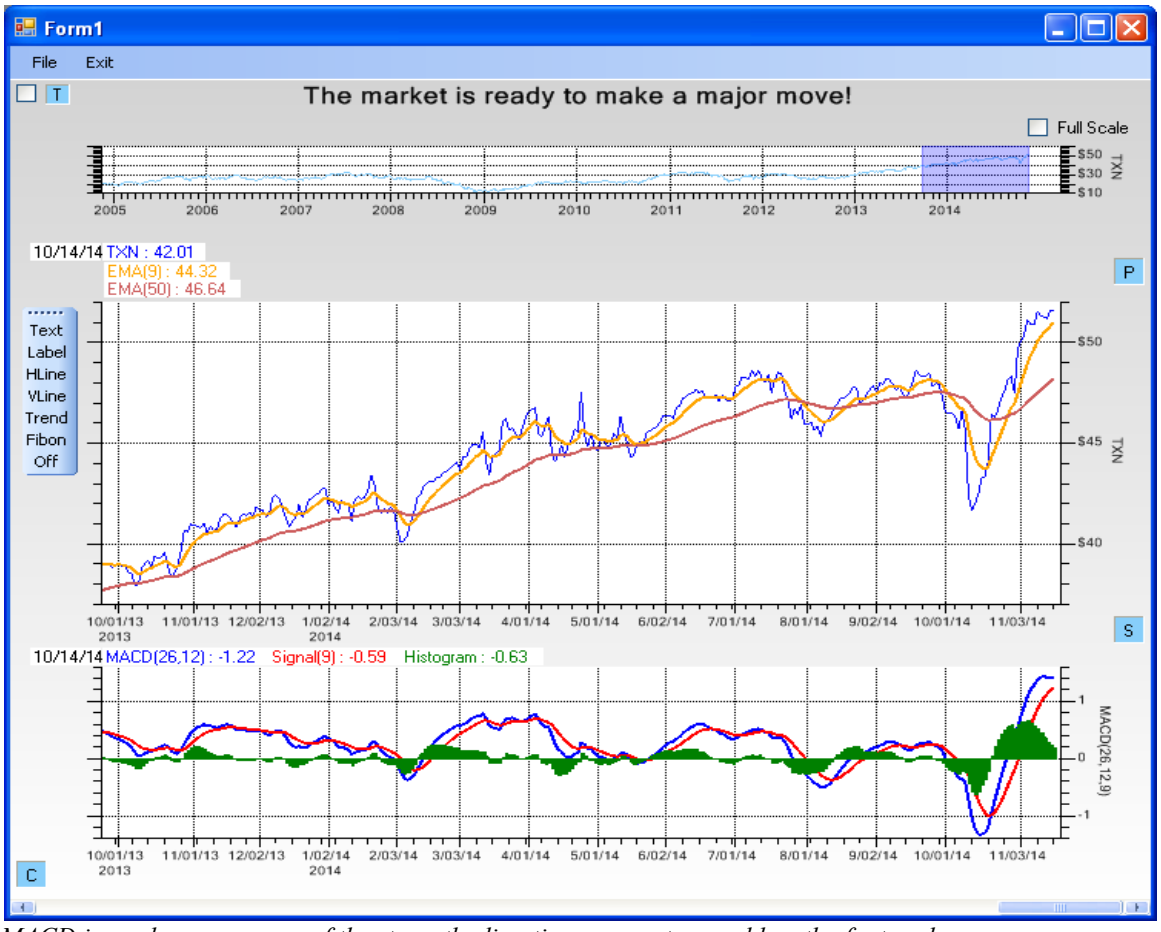
Formulas

The Williams %R indicator is calculated as:

$$\%R[i] = -100 * ((Close[i] - H14[i])/(H14[i] - L14[i]))$$

where Close[i] is the periods closing price, L14[i] is the lowest low for the last N periods, and H14[i] is the highest high for the last N periods, where N is usually 14 periods. This results in a value of -100 to 0. A %R[i] value of over -20 means that the closing price is very near the top of the OHLC range, and a value of under -80 means that the closing price is near the bottom of the OHLC range.

Moving Average Convergence/Divergence (MACD)



MACD is used as a measure of the strength, direction, momentum and length of a trend.

The MACD was developed by Gerald Appel and is discussed in his book, *The Moving Average Convergence Divergence Trading Method*. It is a comparison of several moving averages (exponential) derived from a stocks closing prices.

The first EMA uses a 26 day period and is referred to as the long EMA line. The second EMA uses a 12 day period and is referred to as the short EMA line. The MACD line is calculated by subtracting the long (26) EMA line from the short (12) EMA line. The signal line is calculated as a 9 day EMA of the MACD line. The two lines actually plotted in the indicator are the MACD line, and the signal line. The crossover of MACD and signal lines indicates a buying or selling opportunity. Additionally, a histogram, representing the differences between the MACD line, and the signal line, is usually part of a MACD chart.

Trading Strategies Stochastic

Buy when

Buy when the MACD line passes up and through the signal line - this corresponds to the histogram going positive.

Sell when

Sell when the MACD line falls below the signal line – this corresponds to the histogram going negative.

Formulas

The MACD indicator is calculated as follows:

The long EMA line is calculated as the EMA(*nlong*) of the close prices.

The short EMA line is calculated as the EMA(*nshort*) of the close prices.

The MACD line is calculated as the difference between the long EMA and the short EMA.

The signal line is calculated as the EMA(*nsignal*) of the MACD line.

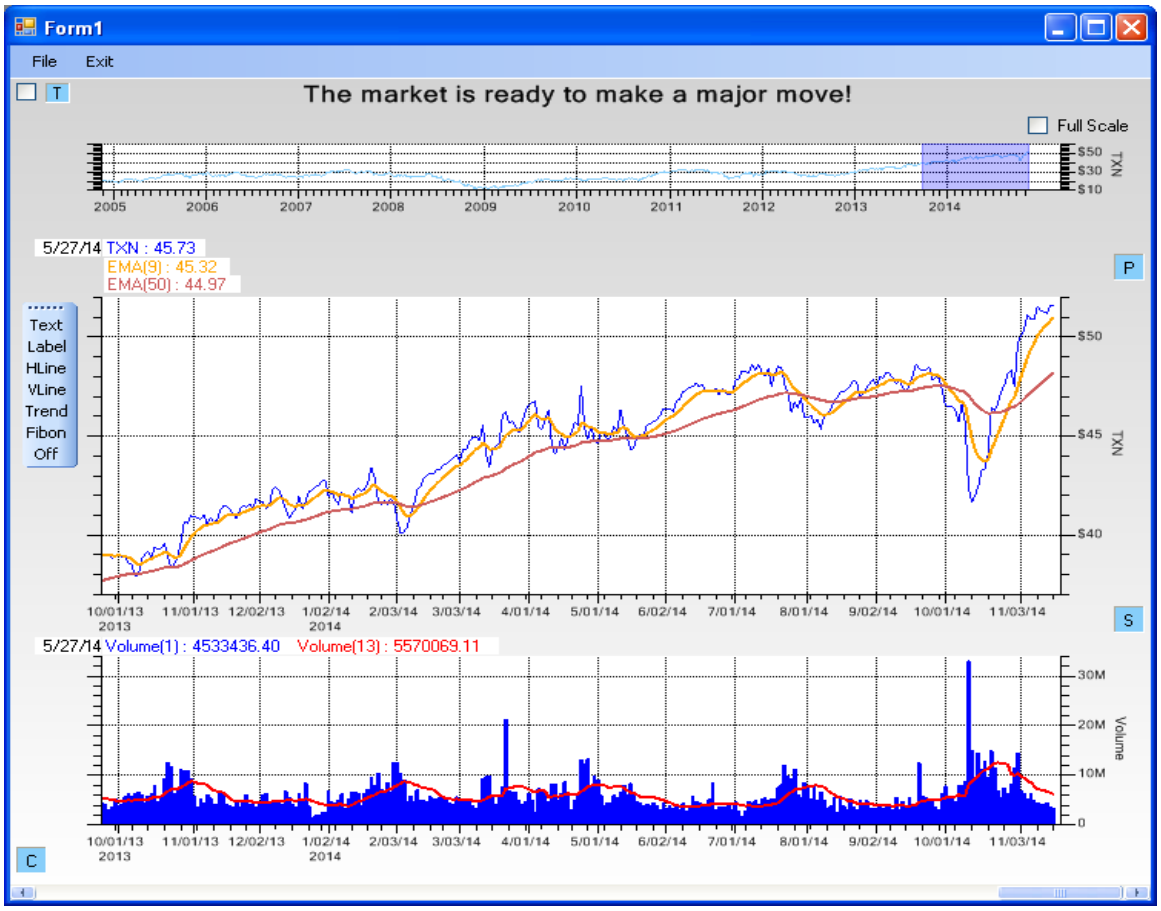
The histogram is calculated as the MACD line minus the signal line.

Where *nlong*, *nshort* and *nsignal* represent the exponential smoothing periods for the long EMA, the short EMA and the signal line, respectively.

Special Note

Some packages use a SMA (Simple Moving Average), rather than an EMA (Exponential Moving Average) in the calculations. That is not how the MACD is normally defined though.

Volume charts



Traders expect that major changes in the direction of a trend are accompanied by high volume.

Volume represents the number of shares, or contracts, which traded during a period of time. Since it is part of the standard OHLCV price packet available in historical data feeds, it gives valuable information about the significance of the price action of a stock. A price movement in either direction is considered more relevant if is is simultaneous with a large increase in share volume. If you are monitoring the trend of a stock, and that stock suddenly reverses the trend, that is the time to check the volume to see if the reversal was accompanied by a sharp increase in the trading volume. If not, then that can be a sign that there is no conviction by the trend reversal, and you can expect the stock to reverse back. Volume charts can be smoothed using the standard smoothing techniques to help filter out the noise of no use to the trader.

Buy when

Buy when the trend moves from negative to positive, if the transition is accompanied by large volume.

Sell when

Sell when the trend moves from positive to negative, if the transition is accompanied by large volume.

Other Chart Features

Event-based Coordinate System

This software makes use of a new set of classes have been added to the underlying QCChart2D charting package in support of an event-based coordinate system. In event-based plotting, the coordinate system is scaled to the number of event objects. Each event object represents an x-value, and one or more y-values. The x-value can be time based, or numeric based, while the y-values are numeric based. Since an event object can represent one or more y-values for a single x-value, it can be used as the source for simple plot types (simple line plot, simple bar plot, simple scatter plot, simple line marker plot) and group plot types (open-high-low-close plots, candlestick plots, group bars, stacked bars, etc.). The most common use for event-based plotting will be for displaying time-based data which is discontinuous: financial markets data for example. In financial markets, the number trading hours in a day may change, and the actual trading days. Weekends, holidays, and unused portions of the day can be excluded from the plot scale, producing continuous plots of discontinuous data.

An event-based coordinate system plots data points equally spaced, regardless of the time stamp. Data plots smoothly transition off-hours, weekends and holidays.

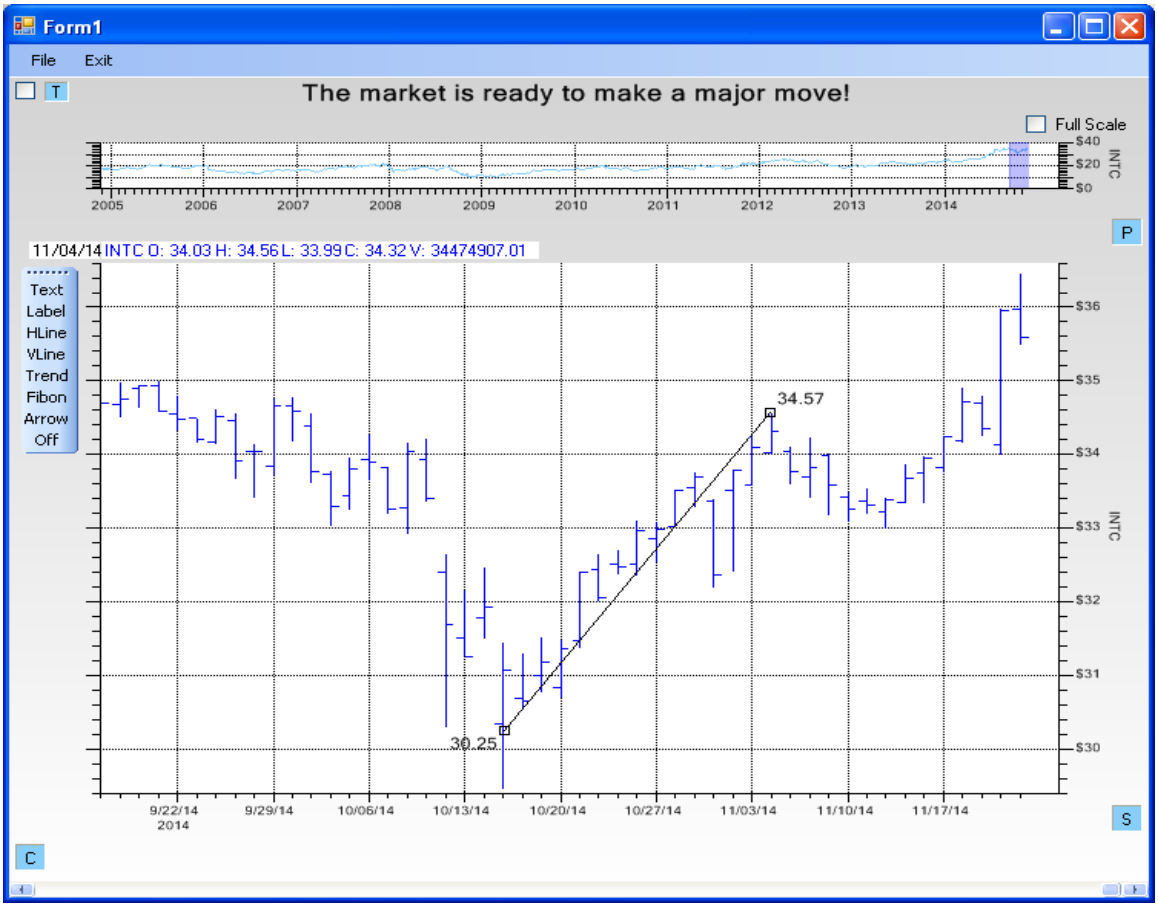
Above is a plot of the OHLC data for INTC (Intel) over the December 2013 Christmas holiday. Note that the only dates included are the trading dates (12/23, 12/24, 12/26, 12/27, 12/30, 12/31, 1 /2 /2014, etc. The non-trading days of 12/25, 12/28, 12/29, 1/1/2104 are not included in the charts coordinate system. The coordinate system is defined by the data contained within. If for some reason you did have data for 12/25/2104, if it was included in the OHLC

data source, it would be included in the scale.

Financial Chart objects which can be dropped into a chart

There is a small set of technical analysis objects which can be placed in the Primary chart. These are trend lines (FinTrendLine), a financial Fibonacci object (FinFibonacciPlot), horizontal (FinHLine) and vertical (FinVLine) data markers, arrows (FinArrow), and two types of labels for annotations. The first label type (FinText) is positioned using normalized coordinates, and does not scroll when the chart scrolls. The second label type (FinLabel) is specified using physical coordinates and will scroll when the chart is scrolled.

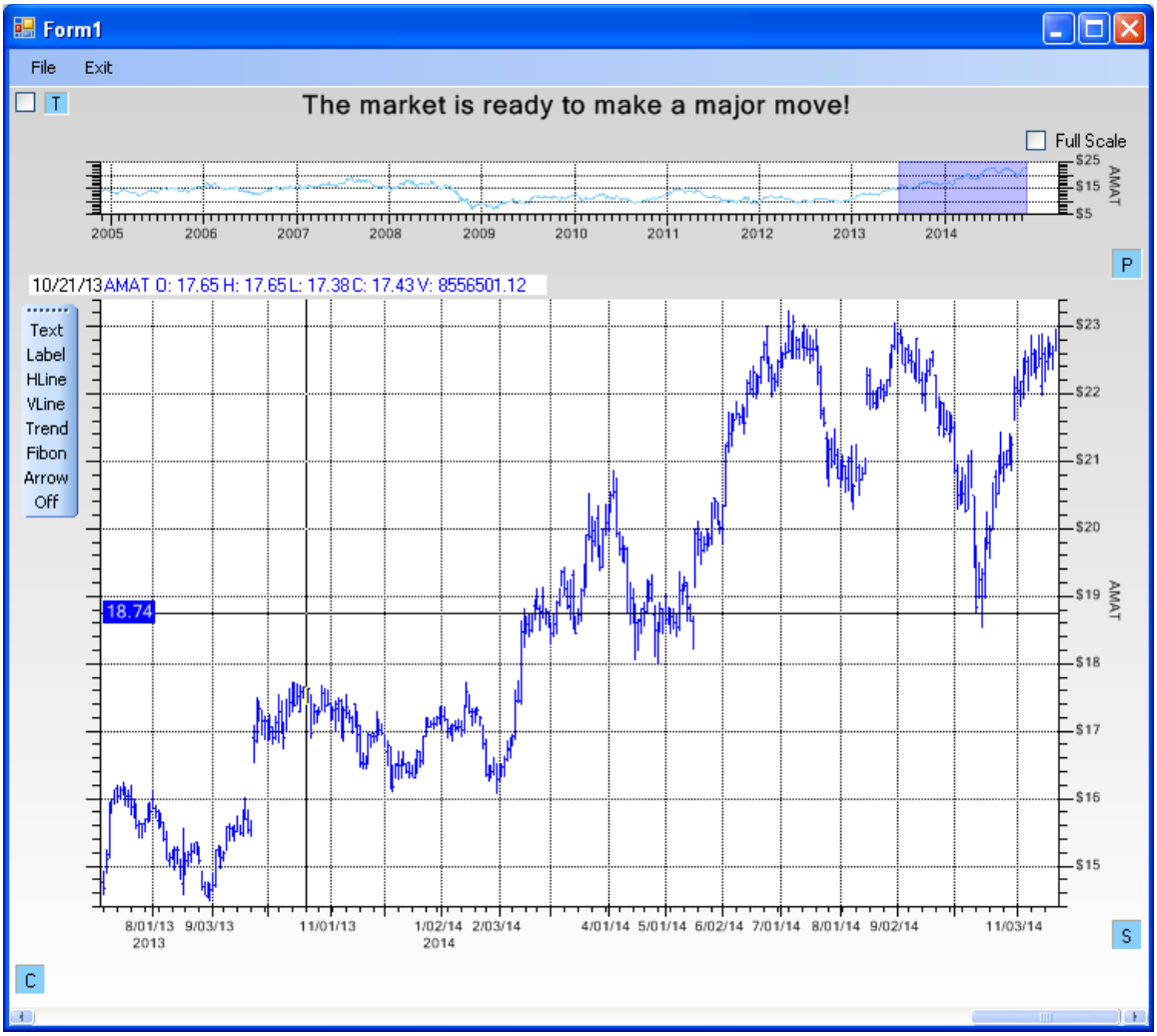
Trend Line



Drawing a trend line helps determine support and resistance levels for a stock trending upwards or downwards.

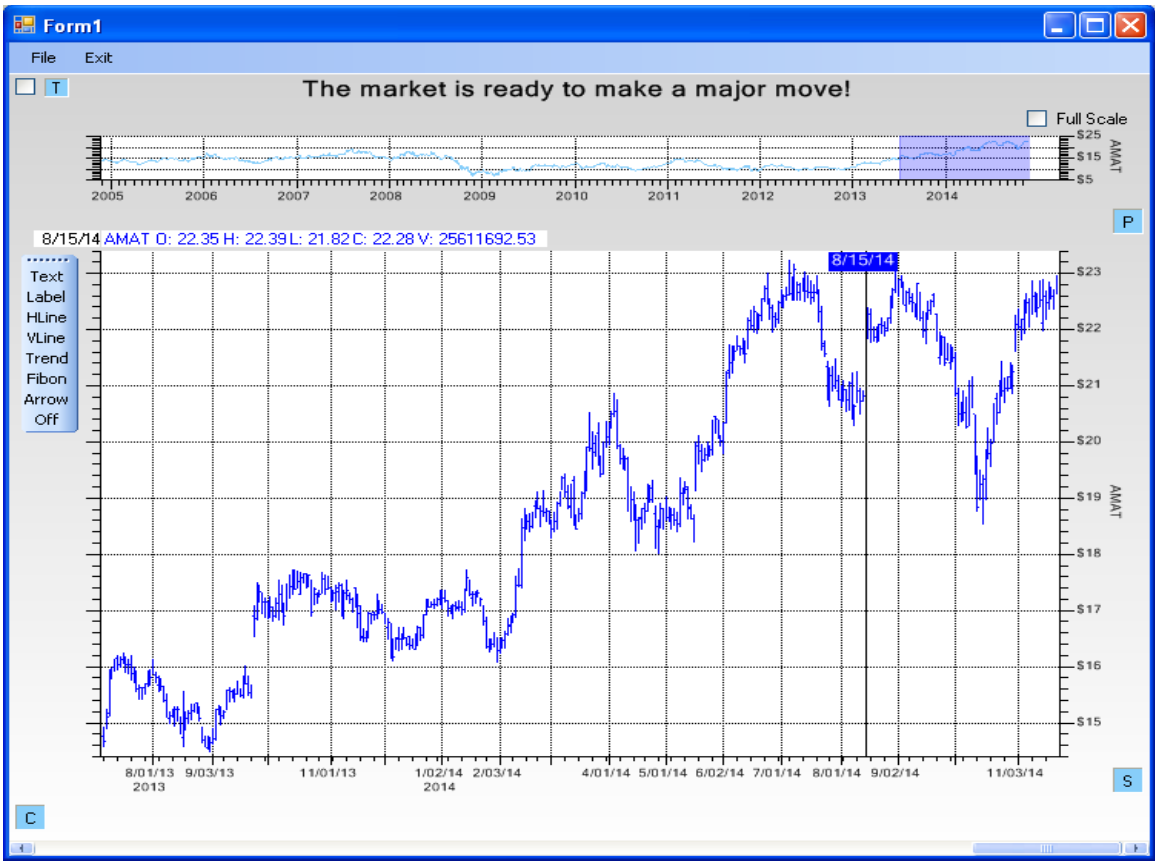
Select the Trend option from the toolbar, then click the start and ending point of the trend line in the Primary Chart window. The trend line can be moved by click-dragging the center of the line. The slope of the line is controlled by click-dragging the endpoints of the trend line. The price values of the trend line endpoints are also displayed. Once placed, you can still adjust the position of the line with great precision.

Horizontal and Vertical data markers



The horizontal data marker is useful for marking support and resistance levels.

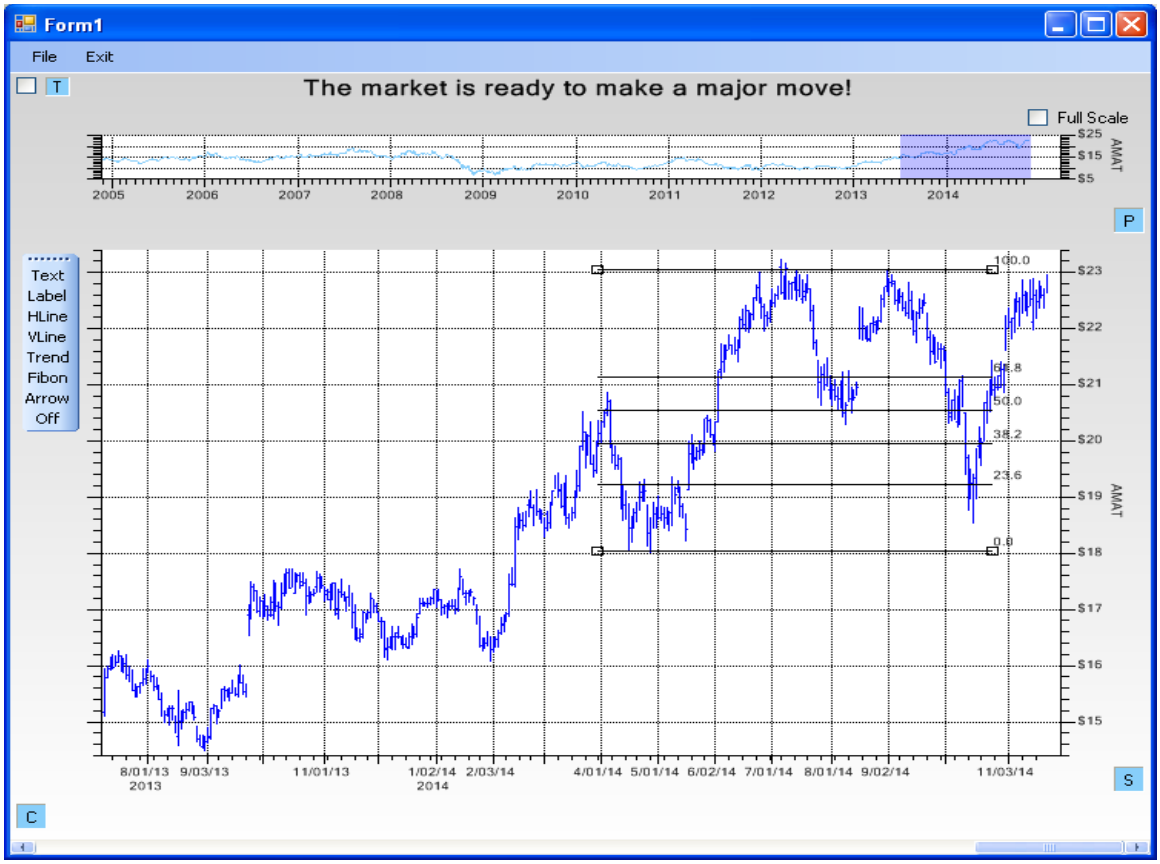
Select the Hline option from the toolbar, click on the chart at the desired location, and a horizontal data marker will appear. It can be moved in a vertical plane. The y-value of the marker is displayed on the left.



A vertical data marker is useful for making important time period.

Similarly, the VLine option will place a vertical data marker on the chart, which can be moved in the horizontal plane. The x-value of the marker is displayed at the top.

Fibonacci Overlay



A Fibonacci overlay is used to predict support and resistance levels for a stock.

A Fibonacci overlay is used in some types of technical analysis wave theory to identify support and resistance levels for a stock. The values on the right of the horizontal lines in the example above (0, 23.6, 38.2, 50.0, 61.8, 100, 161.8 and 261.8) represent percentage values and are calculated using the famous Fibonacci sequence (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233). How do the first set of values derive from the second you ask?

Choose a seed value from the Fibonacci sequence of 89.

233/89 = 2.618

144/89 = 1.618

89/89 = 1.00

55/89 = 0.618

34/89 = 0.382

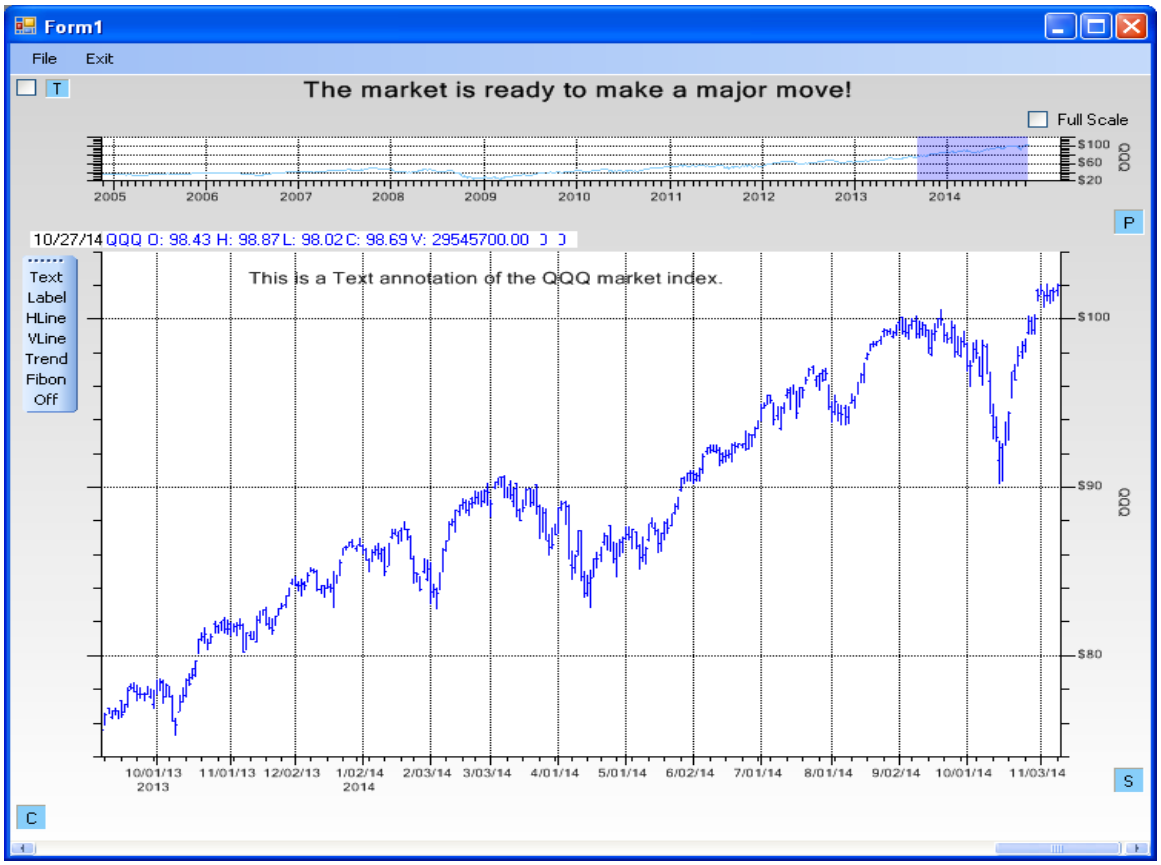
21/89 = 0.236

And the 0.50 level is thrown in for good measure, because market technicians like 50% as a retracement level.

Typically you place the bottom of the Fibonacci rectangle at a local minimum, and the maximum at a local maximum, and the intermediate values represent price support and resistance levels.

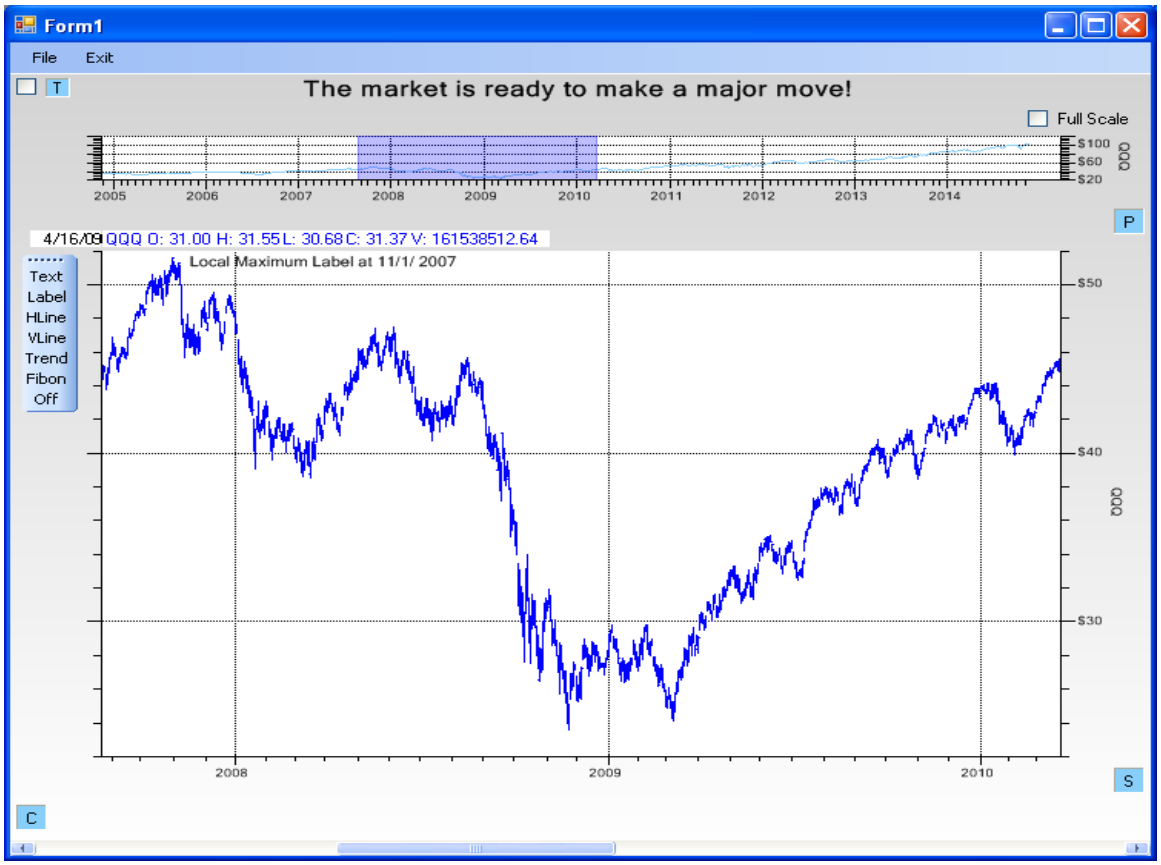
Select the Fibon option from the toolbar, then click the lower-left corner, and the upper-right corner, of where you want the Fibonacci overlay. The Fibonacci overlay will be drawn at the selected location, with the Fibonacci levels 0 and 100 at the click points. The overlay can be adjusted by click-dragging the corners, or the center, of the overlay. If you define the Fibonacci rectangle by clicking the top corner point first, then the bottom corner point, an inverted Fibonacci sequence will be created, with the 0.0 level at the top, and levels increasing towards 262, downward.

Labels for Annotation



FinText objects do not scroll with the chart x-axis, and are used for text that you always want on the chart, regardless of the time frame.

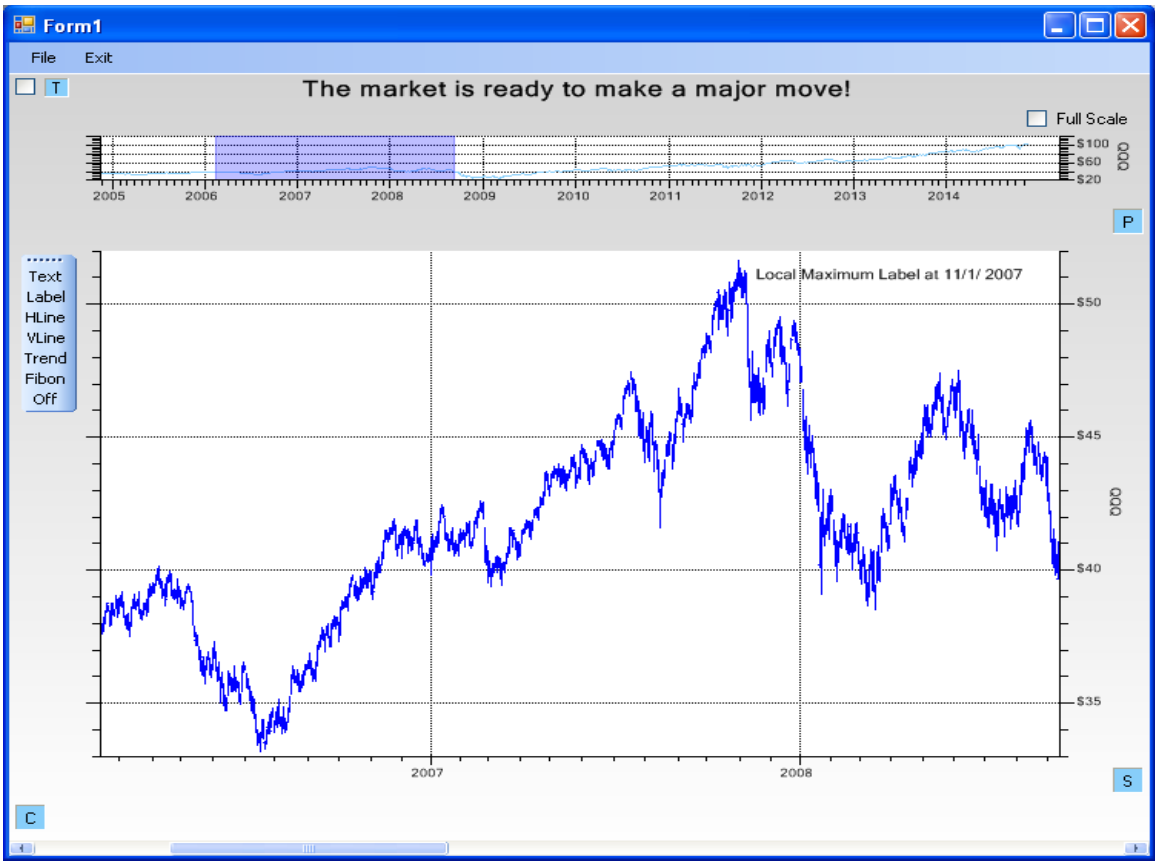
There are two types of labels you can use for annotations. The first uses the FinText class. It is placed in the chart using Normalized graph coordinates. Because of this, it does not scroll, or change position, when the graph y-scale is changed, or when the graph is panned or zoomed. Because Normalized Graph coordinates do not change even if the graphs physical coordinate system changes.



The FinLabel object will scroll with the chart.

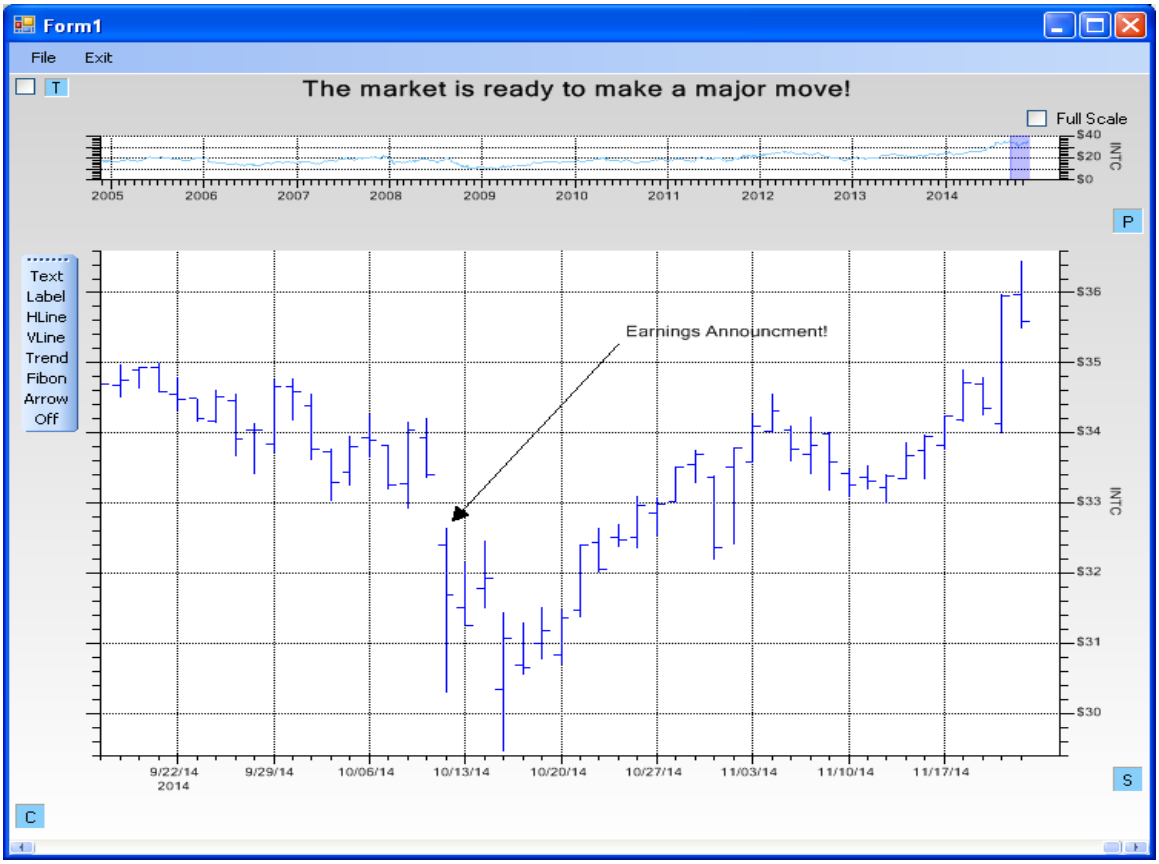
The second type of annotation uses the FinLabel class. It is placed in the chart using Physical graph coordinates. Because of this, it scrolls, and change position, when the graph y-scale is changed, or when the graph is panned or zoomed. Because the FinLabel object sticks to the physical coordinates where it was placed.

Select the Text or Label option from the toolbar, then click the position where you want the label. You will then see a dialog box with options regarding Font, the text string, colors, and justification. When you close the dialog, the label will be drawn at the click location. The overlay can be adjusted by click-dragging the center of the label.



Unlike the FinText object, FinLabel objects do scroll when the x-axis scrolls, and are used when you want the label to track the underlying plot.

Arrows for Annotation

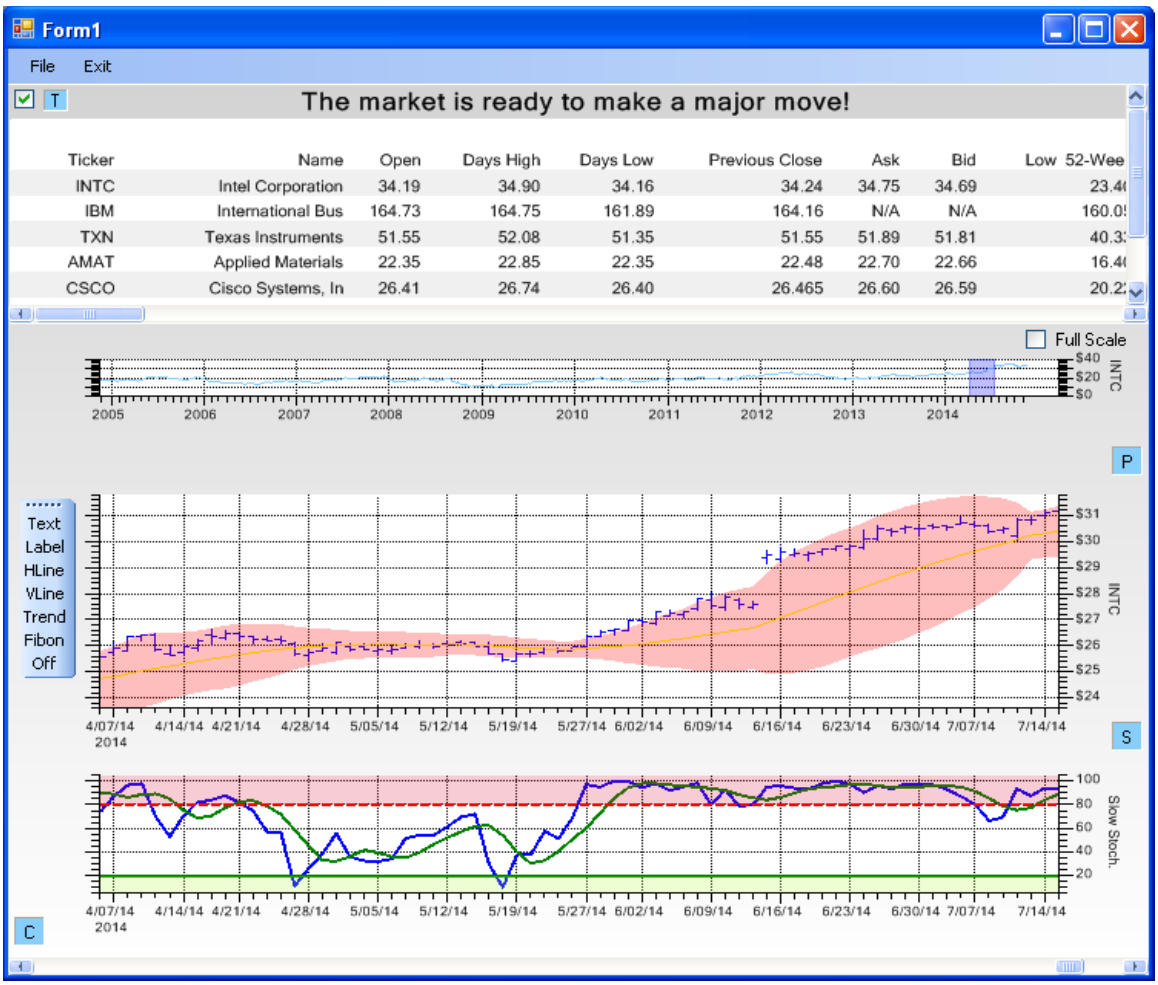


Arrow objects include a text annotation (similar to a FinLabel object). Arrow objects will scroll with the data.

The FinArrow object is a line, ending in an arrowhead at one end, and annotation at the other, which can be place in the primary chart. The line placement and slope, the arrowhead size, the line width, text and font are all user programmable.

Select the Arrow option from the toolbar, then click the starting (the arrow head), and ending position where you want the arrow. You will then see a dialog box with options regarding Font, the text string, colors, justification, and arrowhead size. When you close the dialog, the arrow, and accompanying label will be drawn using the starting and ending click locations. The arrow can be adjusted by click-dragging the endpoints, or the shaft of the arrow.

Current Financial Information Data Table



A Yahoo-based data table is shown at the top of the chart, above the zoom window.

The FinChartView window has the option to display a scrollable table which contains current financial data for a user-defined portfolio of stocks. The user can click on a row and automatically have all of the charts updated to reflect technical indicators for the selected stock. There are two sources for current financial data: Yahoo Fiance, and Quandl. The data they provide is similar, yet different. You can select which stocks (rows) and which financial data items (columns) you want displayed in the table. The table will scroll horizontally and vertically as needed.

3. Class Architecture of the Technical Analysis Charting Tools for .Net Class Library

This chapter presents an overview of the **QCTAChart** class architecture. It discusses the major design considerations of the architecture.

Charting Tools for .Net Class Summary

The **QCTAChart** library is a super set of the **QCChart2D** library. The classes of the **QCChart2D** library are an integral part of the software. A summary of the **QCChart2D** classes appears below.

QCChart2D Class Summary

Chart view class	The ChartView 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, line style, line thickness, etc.) for a chart object.
Auto-Scale classes	The coordinate transform classes use the auto-scale classes to establish the minimum and maximum values used to scale a 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.
File and printer rendering	These classes render the chart image to a printer, to a variety of file formats including JPEG, and BMP, or to a .Net Image object.

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

For each of these categories see the associated description in the **QCChart2D** manual (QCChart2DNetManual.pdf). The **QCTAChart** classes are in addition to the ones above. They are summarized below.

Technical Analysis Charting Tools for .Net Class Summary

The **QCTAChart** classes are a super set of the **QCChart2D** charting software. No attempt should be made to utilize the **QCTAChart** classes without a good understanding of the **QCChart2D** classes. See the **QCChart2DNetManual** PDF file for detailed information about the **QCChart2D** classes.

FinChartView

Com.quinncurtis.chart2dnet.ChartView
FinChartView

The main view class of software, which combines a primary chart, with multiple technical indicators in secondary charts. Also included is a zoom window which controls zooming and panning of all of the charts and a financial data table. It can all be managed by the user from dialog boxes with minimal direct programming. Using it, you can manage a portfolio of securities, from different exchanges, comparing them against one another, and displaying a variety of technical indicators.

FinPlotParameters

FinPlotParameters
- **FinADXIndicatorPlot**
- **FinBollingerBandsPlot**
- **FinCandlestickVolumePlot**
- **FinExponentialMovingAveragePlot**
- **FinMABandsPlot**
- **FinMACDIndicatorPlot**
- **FinMomentumIndicatorPlot**
- **FinMoneyFlowIndicatorPlot**
- **FinParabolicSARPlot**
- **FinPointAndFigureChartPlot**
- **FinRateOfChangeIndicatorPlot**
- **FinRenkoChartPlot**
- **FinRSIIndicatorPlot**
- **FinSimpleMovingAveragePlot**
- **FinStochasticIndicatorPlot**
- **FinTickerItemPlot**
- **FinVolumeAndMAPlot**
- **FinVolumePlot**
- **FinWilliamsRIndicatorPlot**
- **FinZoomPlot**

The FinPlotParameters is the base class for all of financial plots which can appear in the FinChartView windows. It includes the stock plots of the data in the primary chart, the technical indicators, and the zoom plot window.

Primary Window stock plots

FinTickerItemPlot – The FinTickerItemPlot can be configured to plot historical OHLC data as a Line, OHLC, OHLC Bar, Candlestick, Candlestick Volume, or a Mountain plot.

FinPointAndFigureChartPlot – Displays a point and figure chart in the primary window.

FinRenkoChartPlot – Displays a Renko chart in the primary window.

Primary Window Overlay Technical Indicators

FinBollingerBandsPlot - The Bollinger Bands technical indicator, displayed as an overlay in the primary window.

FinExponentialMovingAveragePlot - The Exponential Moving Average technical indicator, displayed as an overlay in the primary window.

FinMABandsPlot - The Moving Average Bands technical indicator, displayed as an overlay in the primary window.

FinSimpleMovingAveragePlot - The Simple Moving Average technical indicator, displayed as an overlay in the primary window.

FinParabolicSARPlot – The Wilder Parabolic SAR technical indicator, displayed as an overlay in the primary window.

Secondary Window(s) Technical Indicators

FinADXIndicatorPlot – The ADX technical indicator, displayed in a secondary window.

FinMACDIndicatorPlot – The MACD technical indicator, displayed in a secondary window.

FinMomentumIndicatorPlot - The Momentum technical indicator, displayed in a secondary window.

FinMoneyFlowIndicatorPlot - The Money Flow technical indicator, displayed in a secondary window.

FinRateOfChangeIndicatorPlot - The Rate of Change technical indicator, displayed in a secondary window.

FinRSIIndicatorPlot - The RSI technical indicator, displayed in a secondary window.

FinStochasticIndicatorPlot - The Stochastic technical indicator, displayed in a secondary window.

FinVolumeAndMAPlot - The Volume and Moving Average of Volume technical indicator, displayed in a secondary window.

FinVolumePlot - The Volume technical indicator, displayed in a secondary window.

FinWilliamsRIndicatorPlot - The Williams R technical indicator, displayed in a secondary window.

Zoom Window

FinZoomPlot - The zoom window, above the primary chart, displays the entirety of the selected stocks historical data. The user can use this chart to zoom in on specific regions of the historical data.

FinDataSourceBase

FinDataSourceBase
- **FinGenericHistoricalDataSource**
  - **FinGoogleHistoricalDataSource**
    - **FinGoogleURLHistoricalDataSource**
    - **FinGoogleCSVFileHistoricalDataSource**
    - **FinGoogleURLIntradayDataSource**
  - **FinMetaStockHistoricalDataSource**
    - **FinMetaStockCSVFileHistoricalDataSource**
    - **FinMetaStockURLHistoricalDataSource**
  - **FinQuandlHistoricalDataSource**
    - **FinQuandlCSVFileHistoricalDataSource**
    - **FinQuandlURLHistoricalDataSource**
  - **FinYahooHistoricalDataSource**
    - **FinYahooCSVFileHistoricalDataSource**
    - **FinYahooURLHistoricalDataSource**
    - **FinYahooURLIntradayDataSource**
- **FinGenericCurrentDataSource**
  - **FinQuandlCurrentDataSource**
    - **FinQuandlURLCurrentDataDataSource**
  - **FinYahooCurrentDataSource**
    - **FinYahooURLCurrentDataSource**

The historical data source classes pull OHLCV historical (daily) data from free URL data sources such as Yahoo, Google, and Quandl. In the Metastock data sources, it supports reading Metastock CSV OHLCV data from files. Both Yahoo, and Google provide intra-day data feeds in addition to the daily historical data feeds.

Current financial data can be read from Yahoo and Quandl URL's. Current financial data consists of current P/E ratio, Yield, Price to Sales ratio, Dept to Equity Ratio, etc. Yahoo and Quandl support similar, yet different sets of current financial data, and each has to be configured uniquely.

FinGenericHistoricalDataSource – The base class for all historical datasource classes

FinGoogleHistoricalDataSource – The base class for Google historical data feeds. Google has officially deprecated all of their Finance data feeds. So while they continue to work, they will probably stop working at some time in the future.

FinGoogleURLHistoricalDataSource – Reads daily historical OHLCV data from the Google Finance URL

FinGoogleCSVFileHistoricalDataSource -Reads historical OHLCV data from Google data file format.

FinGoogleURLIntradayDataSource – Reads intra-day OHLC data from the Google Finance URL

FinMetaStockHistoricalDataSource – The base class for Metastock historical data feeds. The Metastock 7-column and 8-column CSV formats are the most common file formats for historical stock data.

FinMetaStockCSVFileHistoricalDataSource– Reads historical OHLCV data from any file formatted using the Metastock 7 column and 8 column CSV file formats.

FinQuandlHistoricalDataSource – The base class for Quandl historical data feeds

FinQuandlCSVFileHistoricalDataSource - Reads daily historical OHLCV data from a CSV file of Quandl sourced data.

FinQuandlURLHistoricalDataSource - Reads daily historical OHLCV data from the Quandl Finance URL. CSV, XML and JSON data formats are supported.

FinYahooHistoricalDataSource

FinYahooCSVFileHistoricalDataSource - Reads historical OHLCV data from Yahoo data file format.

FinYahooURLHistoricalDataSource - Reads daily historical OHLCV data from the Yahoo Finance URL. CSV, XML and JSON data formats are supported.

FinYahooURLIntradayDataSource – Reads intra-day OHLC data from the Yahoo Finance URL

FinGenericCurrentDataSource – The base class for all current datasource classes

FinQuandlCurrentDataSource -FinQuandlURLCurrentDataDataSource - Reads current financial data from the Quandl URL data source. CSV, XML and JSON data formats are supported.

FinYahooCurrentDataSource - FinYahooURLCurrentDataSource - Reads current financial data from the Yahoo URL data source. CSV, XML and JSON data formats are supported.

Point and Figure Chart Objects

Com.quinncurtis.chart2dnet.LinearAutoScale
FinPointAndFigureAutoScale

Com.quinncurtis.chart2dnet.LinearScale
FinPointAndFigureScale

Com.quinncurtis.chart2dnet.EventCoordinates
FinPointAndFigureCoordinates

Com.quinncurtis.chart2dnet.Axis
FinPointAndFigureYAxis

Com.quinncurtis.chart2dnet..EventAxis
FinPointAndFigureXAxis

Com.quinncurtis.chart2dnet.EventAxisLabels
FinPointAndFigureXAxisLabels

Com.quinncurtis.chart2dnet.GroupPlot
FinPointAndFigurePlot

The Point and Figure charts are unique in many different ways. The x- and y-scales of a Point and Figure coordinate system are unique. The x-scale is non-linear with respect to time, and the y-scale is non-linear and non-logarithmic with respect to stock price. So a unique set of classes specific to Point and Figure charts were created for the software.

FinPointAndFigureAutoScale

Used to auto-scale the plotting area for a Point and Figure chart

FinPointAndFigureScale

A Point and Figure chart uses a unique, non-linear y-scale based on box size. As the y-values increase in the scale from 0.1 to 1000, the box size changes.

FinPointAndFigureCoordinates

The Point and Figure chart uses an event-based coordinate system which is non-linear (PointAndFigureScale) in the y-scale, and linear, but not time-based, or event number-based, in the x-scale.

FinPointAndFigureYAxis

Used in a Point and Figure chart to display the y-axis of the Point and Figure coordinate system.

FinPointAndFigureXAxis

Used in a Point and Figure chart to display the x-axis of the Point and Figure coordinate system

FinPointAndFigureXAxisLabels

Used in a Point and Figure chart to display the x-axis labels of a PointAndFigureXAxis. The PointAndFigureYAxis can use the standard NumericAxisLabels class for display of the axis tick mark values.

FinPointAndFigurePlot

Plots the Point and Figure chart as a collection of Xs (uptrends) and Os (downtrends).

New Plot Objects you can use to plot OHLCV data in ChartView, or FinChartView window

Com.quinncurtis.chart2dnet.GroupPlot
FinCandlestickVolumePlot
FinOHLCBarPlot

FinCandlestickVolumePlot

This OHLCV plot type uses the volume value, in each OHLCV data item, to control the width of the respective candlestick plot item.

FinOHLCBarPlot

This OHLCV plot type uses the displays the OHLC data in a vertical floating bar format.

New Financial Chart Objects you can place in a chart

Com.quinncurtis.chart2dnet.SimplePlot
FinFibonacciPlot

Com.quinncurtis.chart2dnet.Marker
FinHLine
FinVLine

Com.quinncurtis.chart2dnet.SimpleLineMarkerPlot
FinTrendLine
FinArrow

Com.quinncurtis.chart2dnet.StringLabel
FinText
FinLabel

FinFibonacciPlot

Place a set of Fibonacci levels in a chart as an indicator of retracement prices. Object is moveable and resizeable.

FinHLine

Place a horizontal data marker in a chart, displaying the y-axis value of the data marker on the end. Object is moveable.

FinVLine

Place a vertical data marker in a chart, displaying the x-axis value of the marker on the end. Object is moveable.

FinTrendLine

Place a trend line in a chart. The object can be moved whole, and the endpoints can be moved independently.

FinText

Place a text label in a chart using Normalized Graph Coordinates, which does not move when the chart scale changes.

FinLabel

Place a text label in a chart using Physical Graph Coordinates, which will follow the data as the chart is panned and zoomed.

FinArrow

Place an arrow in a chart. The arrow can be moved whole, and the endpoints can be moved independently. The arrow can have a text annotation at the tail end of the arrow.

Other useful classes

FinStrings

All of the strings used in the software are defined in a Dictionary object in the static FinStrings class.

FinChartConstants

Consolidates most of the constants used throughout the software

FinChartData

Organizes all of the historical and current financial datasets used in FinChartView window.

FinChartPlotBase

Contains all of the charting objects needed by a chart in FinChartView window. Each chart (Primary chart, secondary charts, and zoom chart) is its own FinChartPlotBase object.

FinChartSupport

A grouping of common support routines for the software.

FinTimeSeriesDataset

Subclasses a EventGroupDataset and adds a historical data source object which will acquire the specified OHLCV data from from a URL or a file

FinTechPrimaryChartDialog

The dialog box for the Primary chart options.

FinTechSecondaryChartDialog

The dialog box for the Secondary chart options.

FinEditAttributeDialog

Edit a single line attribute with color, line style, line thickness and alpha value options

FinEditAttributeDialog2

Edit one to three attributes with color, line style, and line thickness options

FinEditAttributeDialog3

Edit one to three attributes with color, line style, and line thickness options. The first attribute also includes an alpha option.

FinEditAttributeDialog4

Edit a group of line and fill attributes for a single object. Includes line color, fill color, line style, line thickness, line alpha, fill alpha and bar width options.

FinEditChartAttributes

Edit the general characteristics of the primary and secondary charts. Includes the chart size, the button colors, background colors, plot area style and colors, axes line and text colors, font sizes, and zoom windows, data table and default date format (US MM/DD/YY format or Euro DD/MM/YY format).

FinEditDataTableDialog

Edit the data table characteristics. Includes the number of rows and columns, the table font and color, and the background colors and style.

FinEditFillColorsDialog

Edit one to four fill attributes for an objects which use multiple fill colors. The first fill color also has an alpha option.

FinEditLimitsAttributesDialog

Edit the attributes for the high and low limits used in technical indicator charts. It includes unique colors for the high and low limits, and common values for r the line style, line thickness, fill mode and fill alpha value.

FinEditLineDialog

Edit a line object which also has text, date, or numeric readouts, such as the FinTrendLine, FinHLine and FinVLine and FinArrow. Line attributres include the line color, line style, line thickness and grab handles attributes. Text attributes include general Font options, and numeric precision.

FinEditPointAndFigureDialog

Edit the characteristics of a Point and Figure plot. Includes the upside and downside color attributes, box size mode, box size value (if fixed), pricing mode, reversal count and plot type.

FinEditRenkoDialog

Edit the characteristics of a Renko plot. Includes the upside and downside color attributes, box size mode, box size value (if fixed), pricing mode and reversal count.

FinEditScatterPlotAttributeDialog

Edit the characteristics of a scatter plot. Includes the color, symbol, symbol size, line width, symbol fill and alpha value.

FinEditTextDialog

Edit the characteristics of text object. Includes the text Font, text string, text color, background color options, clipping options, vertical and horizontal justification options.

4. QCChart2D for *.Net*

This chapter is a summary of the information in the **QCChart2DNetManual** PDF file. It is not meant to replace that information. Refer to that manual for detailed information concerning these classes.

QCChart2D for *.Net* Class Summary

The following categories of classes realize these design considerations.

Chart view class	The chart view class is a **UserControl** subclass that manages the graph objects placed in the graph
Data classes	There are data classes for simple xy 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, line style, line thickness, etc.) for a chart object.
Auto-Scale classes	The coordinate transform classes use the auto-scale classes to establish the minimum and maximum values used to scale a 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.
File and printer rendering	These classes render the chart image to a printer, to a variety of file formats including JPEG, and BMP, or to a .Net **Image** object.
Miscellaneous utility classes	Other classes use these for data storage, file I/O, and data processing.

A summary of each category appears in the following section.

Chart Window Classes

System.Windows.Forms.UserControl
    ChartView

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

Data Classes

ChartDataset
    SimpleDataset
        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 is initialized using **ChartCalendar** dates (a wrapper around the System.DateTime value class) in place of the x- or 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 device coordinates.
WorldCoordinates	This class derives from the **UserCoordinates** class and maps a device independent world coordinate system on top of the .Net device coordinate

system.

WorkingCoordinates

This class derives from the **WorldCoordinates** class and extends the physical coordinate system of the *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
ChartGradient

This class consolidates the common line and fill attributes as a single class. Most of the graph objects have a property of this class that controls the color, line thickness and fill attributes of the object. The **ChartGradient** class expands the number of color options available in the **ChartAttribute** class.

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

ChartGradient A **ChartGradient** can be added to a **ChartAttribute** object, defining a multicolor gradient that is applied wherever the color fill attribute is normally used

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, line style, and gradient 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

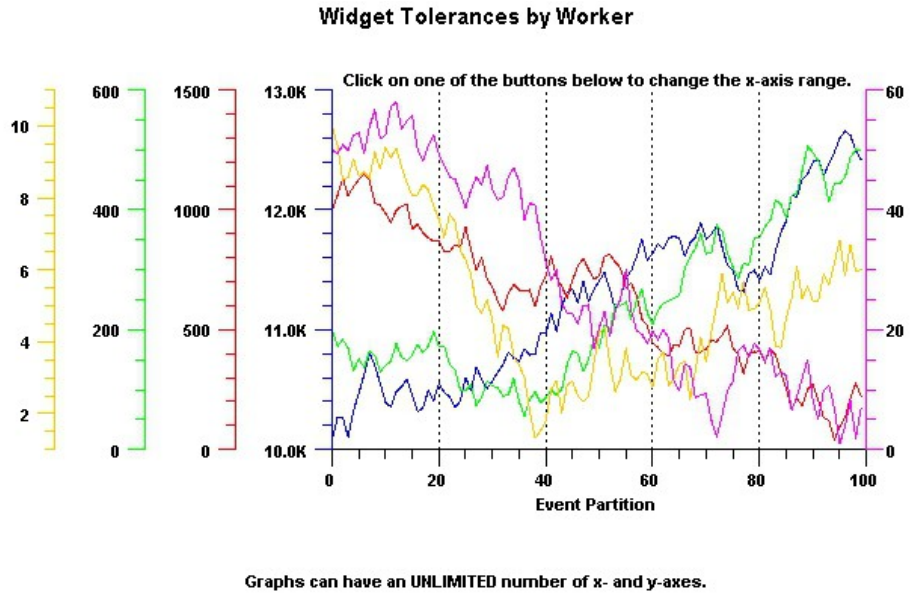
information common to all chart objects. This class includes references to instances of the **ChartAttribute** and **PhysicalCoordinates** classes. The **ChartAttribute** class contains basic color, line style, and gradient information for the object, while the **PhysicalCoordinates** maintains the coordinate 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, a color gradient, or a texture.

Axis Classes

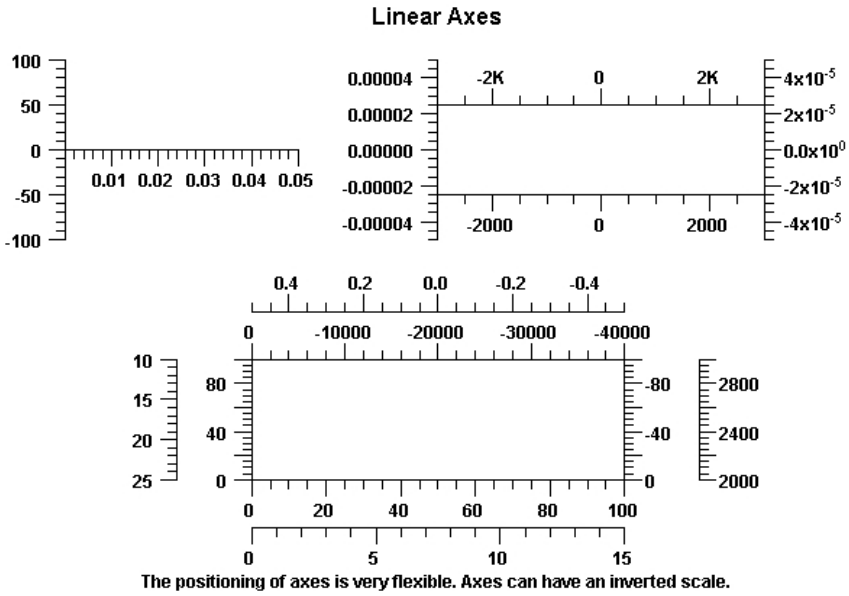
Axis
LinearAxis
PolarAxes
AntennaAxes
ElapsedTimeAxis
LogAxis
TimeAxis

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.



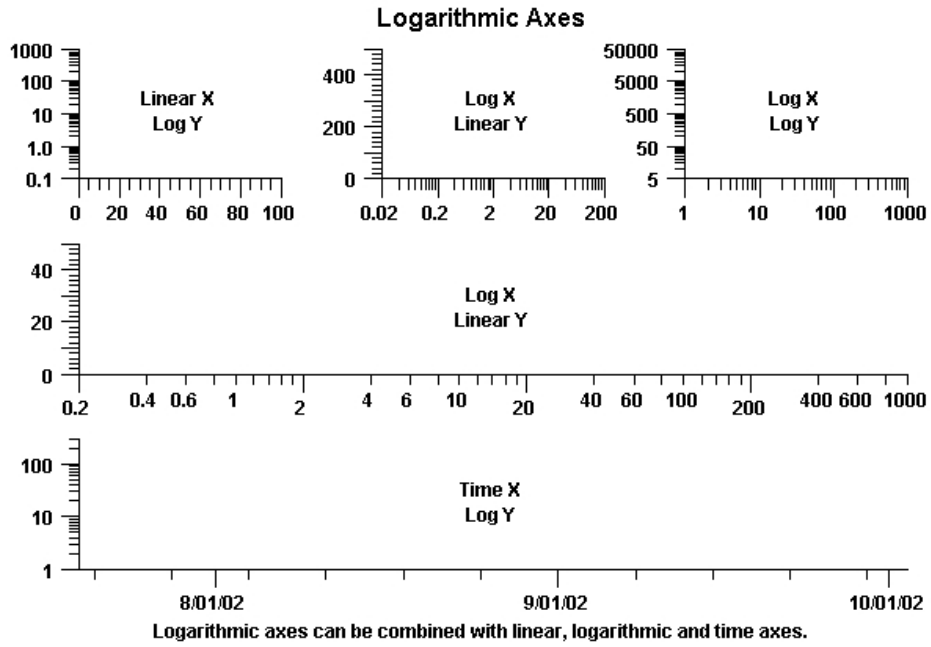
Axis

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



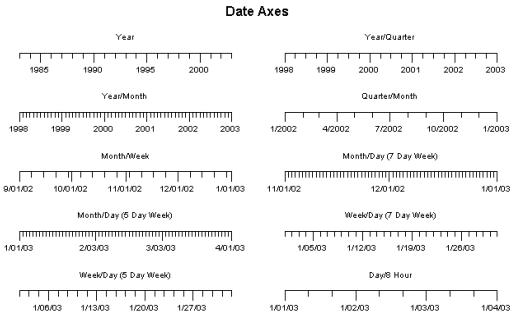
LinearAxis

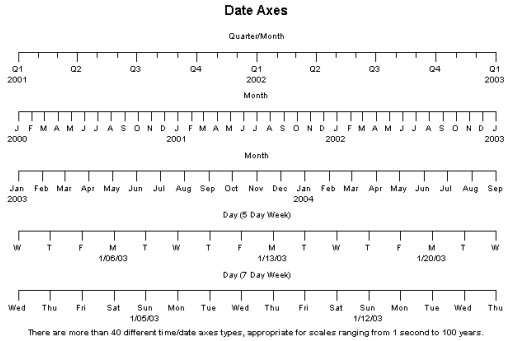
This class implements a linear axis with major and minor tick marks placed at equally spaced intervals.

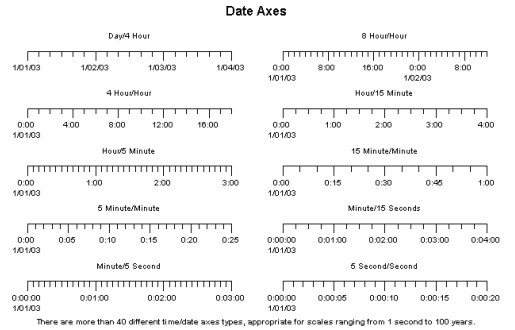


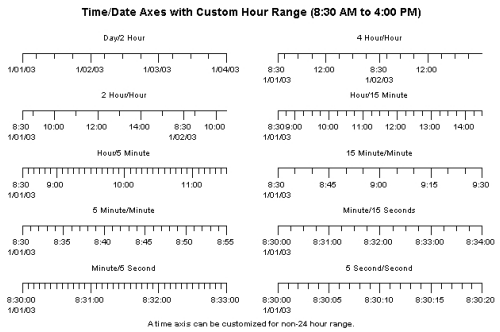
LogAxis

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.



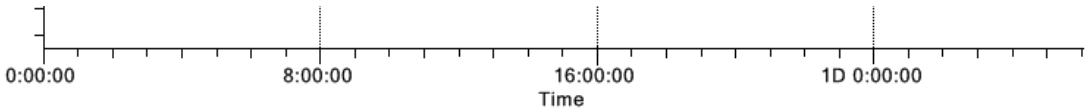






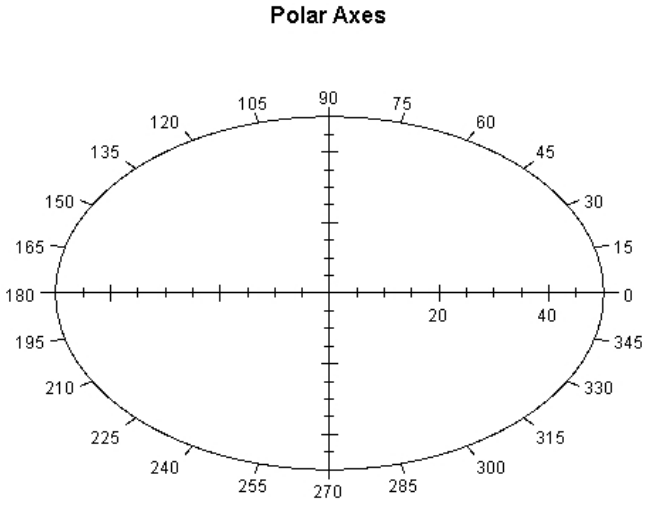
TimeAxis

This class is the most complex of the axis classes. It supports time scales ranging from 1 milliseond to hundreds of years. Dates and times are specified using the .Net **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

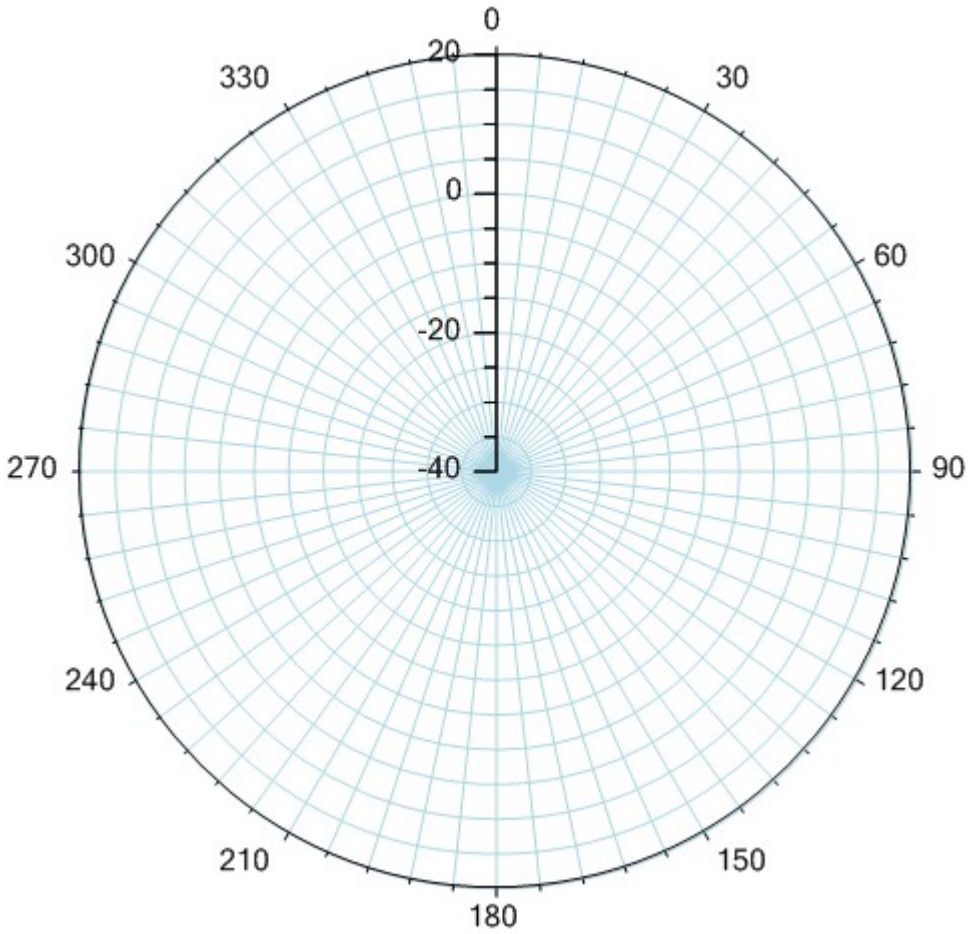
The elapsed time axis is very similar to the linear axis and is subclassed from that class. The main difference is the major and minor tick mark spacing calculated by the CalcAutoAxis method takes into account the base 60 of seconds per minute and minutes per hour, and the base 24 of hours per day. It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



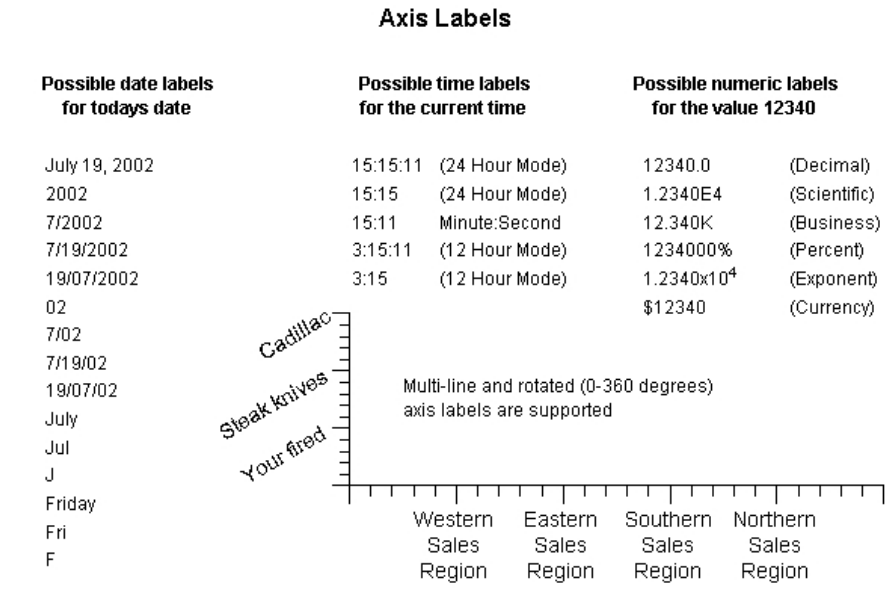
AntennaAxes This class has two axes: one linear y-axis and one circular axis. The linear axis is scaled for the desired range of radius values. This can extend from minus values to plus values. The second axis is a circle centered on the origin with a radius equal to the range of the radius scale. This 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.



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

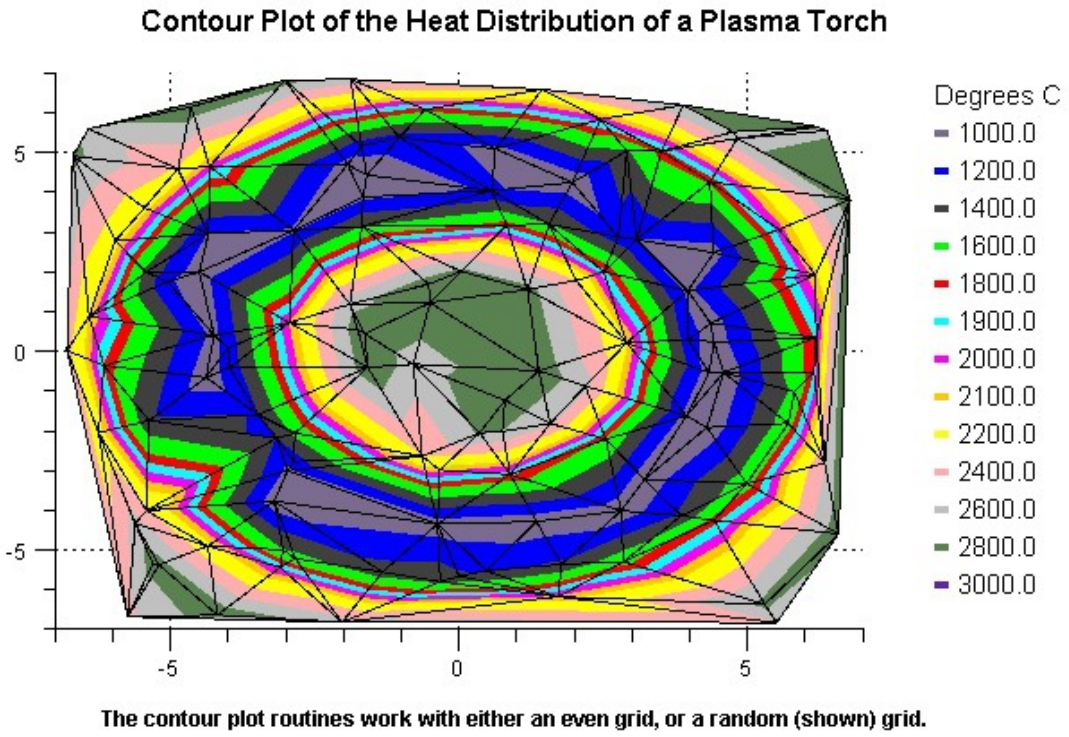
AxisLabels	This class is the abstract base class for all axis label objects. It places numeric labels, date/time labels, or arbitrary text labels, at the major tick marks of the associated axis object. In addition to the standard font options (type, size, style, color, etc.), axis label text can be rotated 360 degrees in one degree increments.
NumericAxisLabels	This class labels the major tick marks of the **LinearAxis**, and **LogAxis** classes. The class supports many predefined and user-definable formats, including numeric, exponent, percentage, business and currency formats.
StringAxisLabels	This class labels the major tick marks of the **LinearAxis**, and **LogAxis** classes using user-defined strings.
TimeAxisLabels	This class labels the major tick marks of the associated **TimeAxis** object. The class supports many time (23:59:59) and date (5/17/2001) formats. It is also possible to define custom date/time formats.
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.

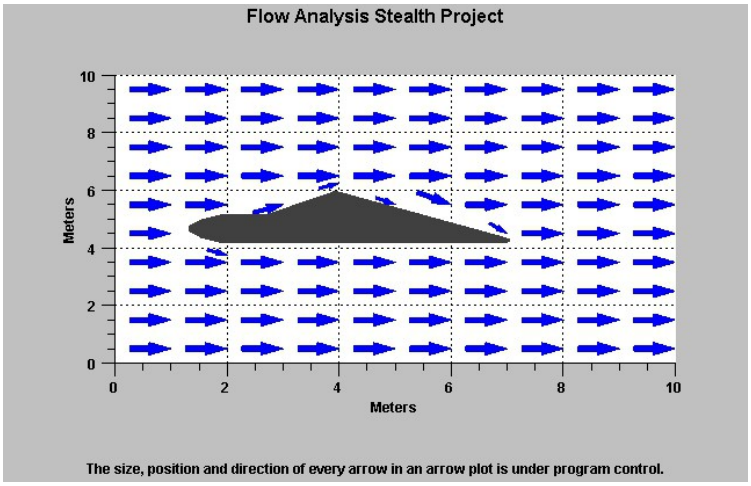
Group Plot Classes

GroupPlot
- **ArrowPlot**
  - **BoxWhiskerPlot**
- **BubblePlot**
- **CandlestickPlot**
  - **CellPlot**
- **ErrorBarPlot**
  - **FloatingBarPlot**
  - **FloatingStackedBarPlot**
  - **GroupBarPlot**
  - **GroupVersaPlot**
- **HistogramPlot**
- **LineGapPlot**
- **MultiLinePlot**
- **OHLCPlot**
  - **StackedBarPlot**
- **StackedLinePlot**
  - **GroupVeraPlot**

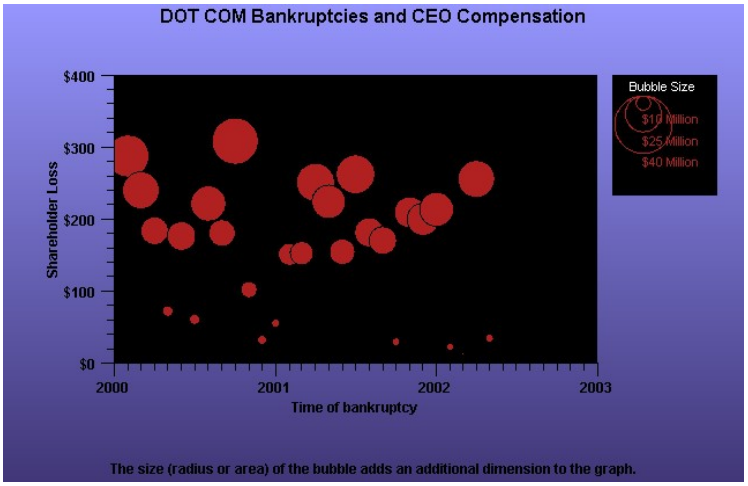
Group plots use data organized as arrays of x- and y-values, where there is one or more y for every x.. Group plot types include multi-line plots, stacked line plots, stacked bar plots, group bar plots, error bar plots, floating bar plots,

floating stacked bar plots, open-high-low-close plots, candlestick plots, arrow plots, histogram plots, cell plots, "box and whisker" plots, and bubble plots.

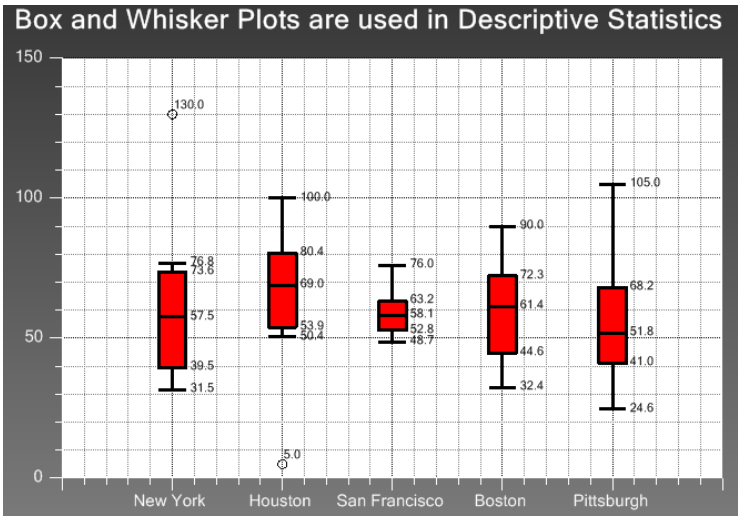
GroupPlot This class is an abstract base class for all group plot classes.



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

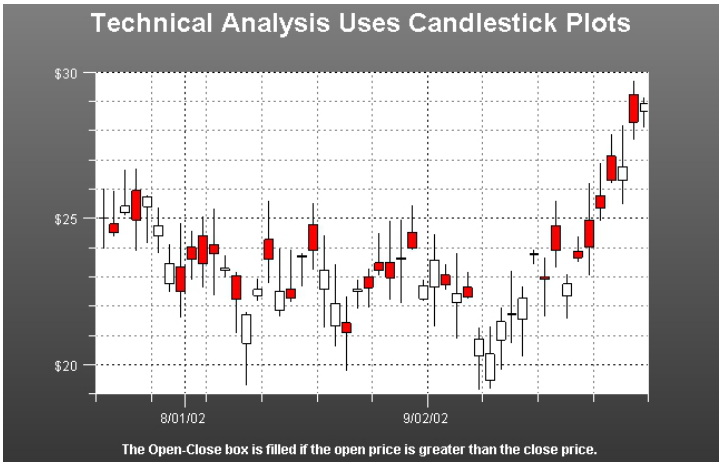


BubblePlot This class is a concrete implementation of the **GroupPlot** class and displays bubble plots. The values in the dataset specify the position and size of each bubble in a bubble chart.



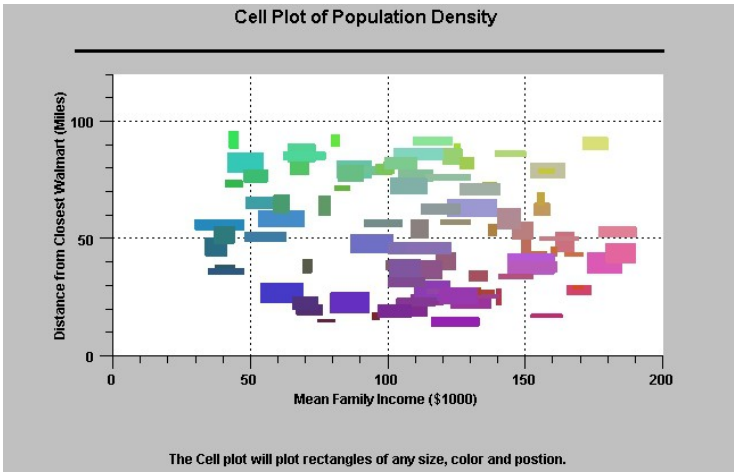
BoxWhiskerPlot

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.

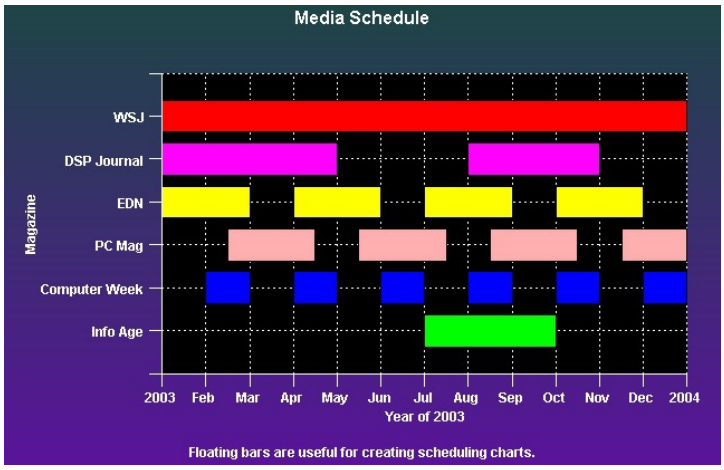


CellPlot

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.

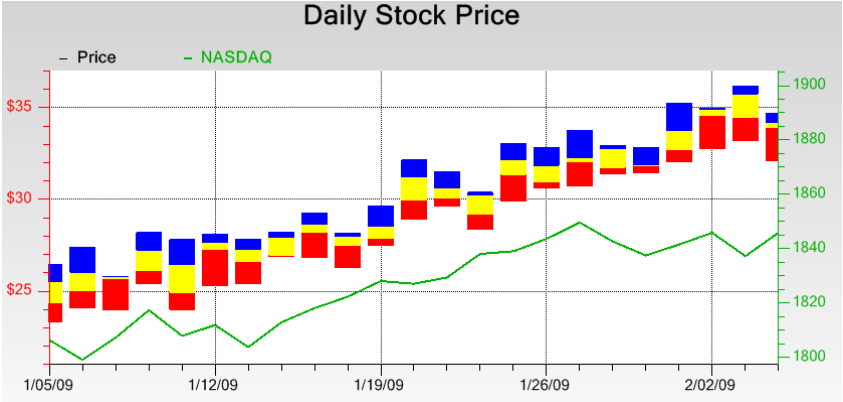
ErrorBarPlot

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

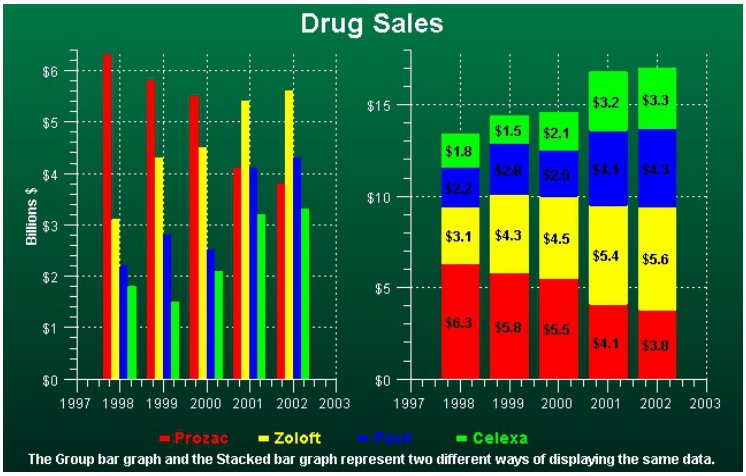


FloatingBarPlot

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.

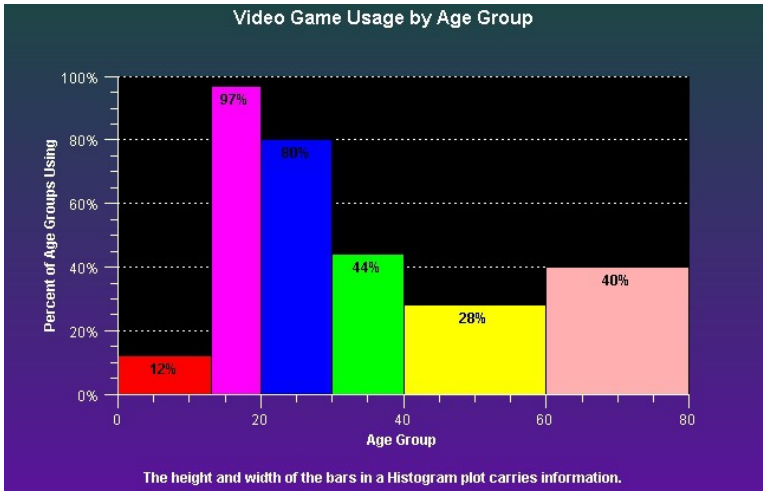


GroupBarPlot This class is a concrete implementation of the **GroupPlot** class and displays group data in a group bar format. Individual bars, the height of which corresponds to the group y-values of the dataset, display side by side, as a group, justified with respect to the x-position value for each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

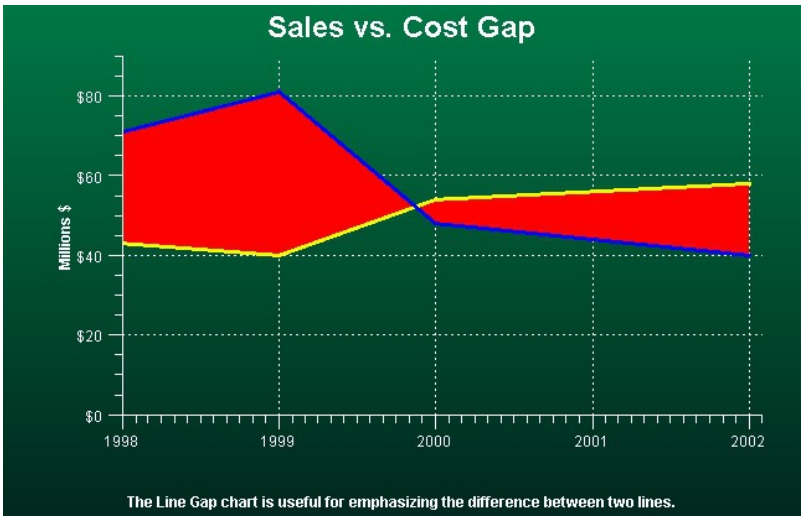
GroupVeraPlot The **GroupVersaPlot** is a plot type that can be any of the eight group plot types: GROUPBAR, STACKEDBAR, CANDLESTICK, OHLC, MULTILINE, STACKEDLINE, FLOATINGBAR and FLOATING_STACKED_BAR. Use it

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.



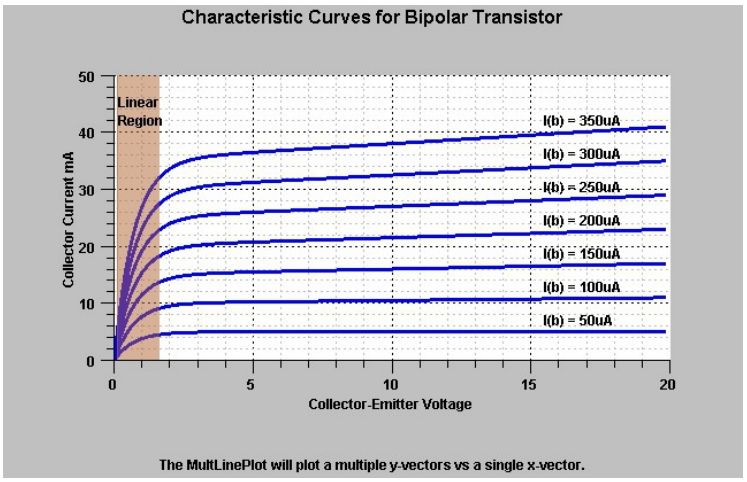
HistogramPlot

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.

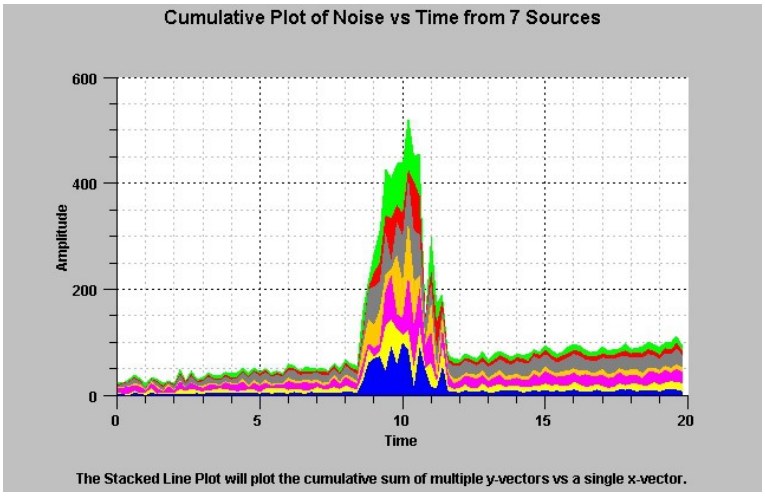


MultiLinePlot

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.



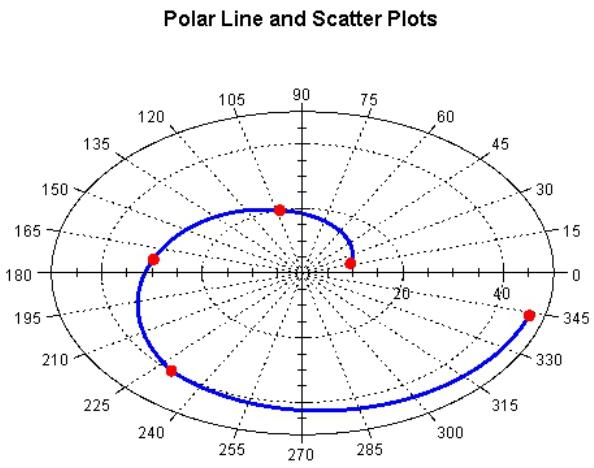
StackedLinePlot This class is a concrete implementation of the **GroupPlot** class and displays data in a stacked line format. In a stacked line plot each group is stacked on top of one another, each group line a cumulative sum of the related group items before it.

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.

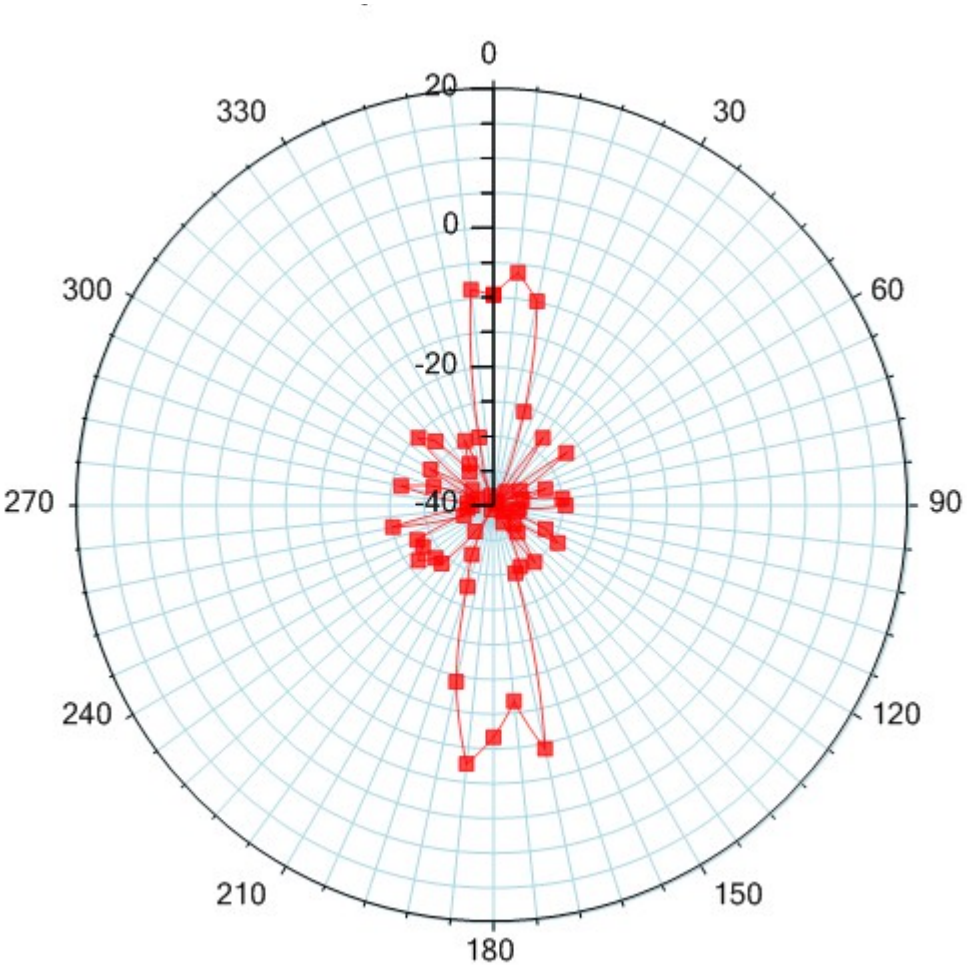
Antenna Plot Classes

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.

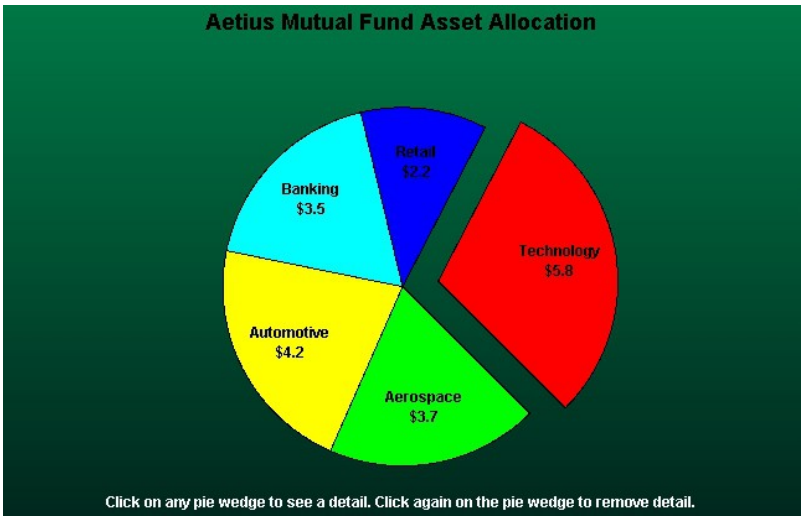


AntennaLineMarkerPlot

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.

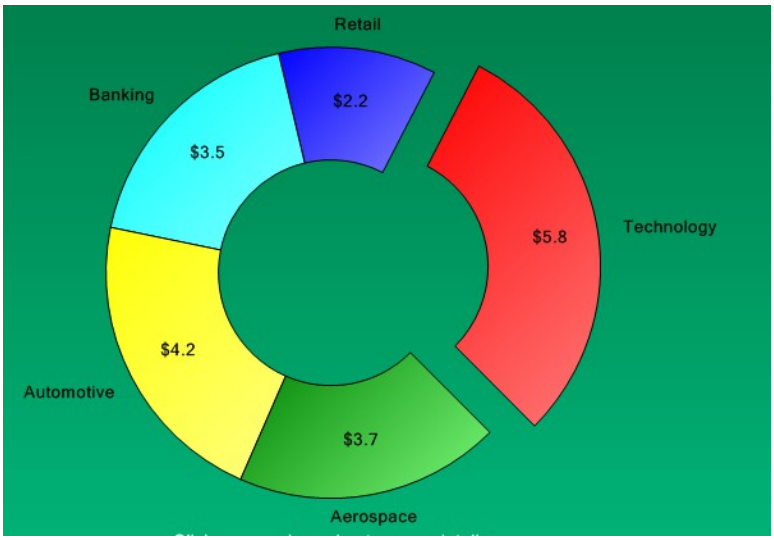
Pie and Ring Chart Classes

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

The pie chart 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

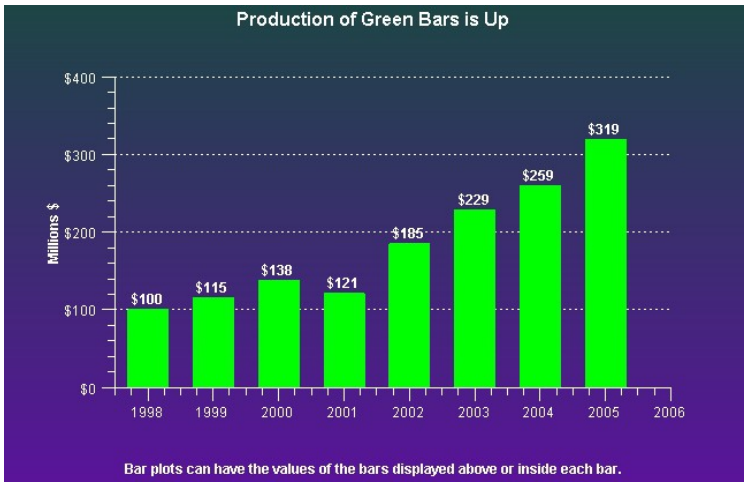
The ring chart plots data in a modified pie chart format known as a ring chartt. It uses data organized as arrays of x- and y-values, where an x-value represents the numeric value of a ring segment, and a y-value specifies the offset (or "explosion") of a ring segment with respect to the origin of the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

Simple Plot Classes

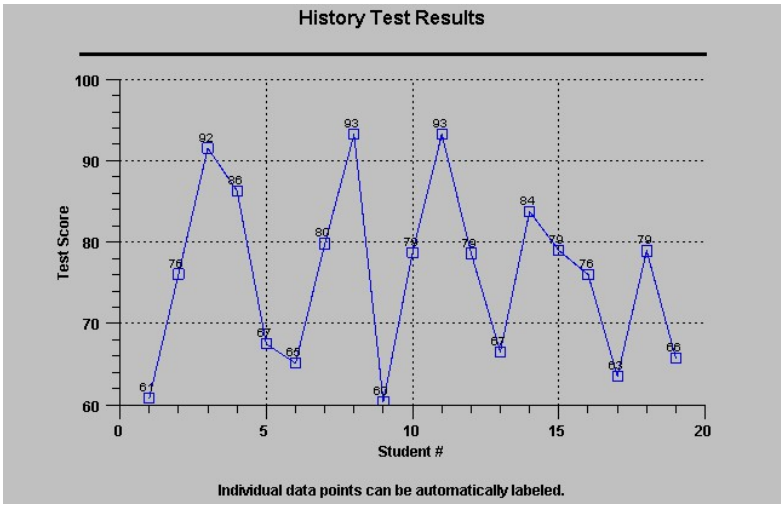
SimplePlot
- **SimpleBarPlot**
- **SimpleLineMarkerPlot**
- **SimpleLinePlot**
- **SimpleScatterPlot**
- **SimpleVeraPlot**

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.

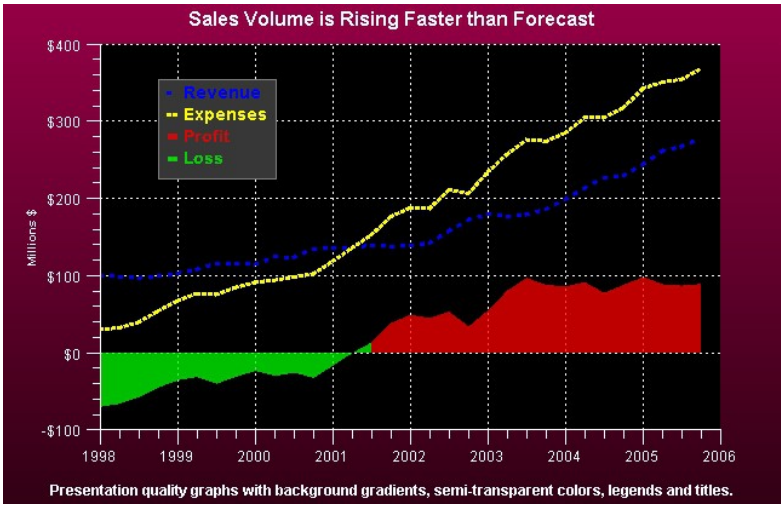


SimpleBarPlot This class is a concrete implementation of the **SimplePlot** class and displays data in a bar format. Individual bars, the maximum value of which corresponds to the y-values of the dataset, are justified with respect to the x-values.



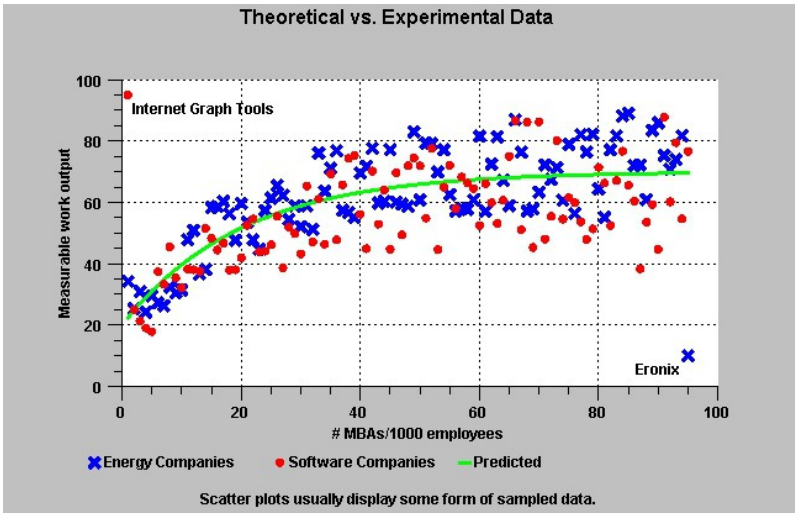
SimpleLineMarkerPlot

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 **System.Drawing.Drawing2D.GraphicsPath** class, placing the shape in a chart using a position defined in chart coordinates. A chart can display any object that can be defined using **System.Drawing.Drawing2D**.GraphicsPath class.
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**
- **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 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.

File and Printer Rendering Classes

ChartPrint
BufferedImage

ChartPrint This class implements printing using the .Net **System.Drawing.Printing** print-related services. It can select, setup, and output a chart to a printer.

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

Miscellaneous Utility Classes

ChartCalendar
CSV
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 Point and PointF classes.
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 Array class, adding routines for resizing of the array, and the insertion and deletion of double based data elements.
DoubleArray2D	This class is used as an alternative to the standard .Net 2D Array class, adding routines for resizing of the array, and the insertion and deletion of double based data elements.
BoolArray	This class is used as an alternative to the standard .Net Array class, adding routines for resizing of the array, and the insertion and deletion of bool based data elements.
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 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 for .*Net* library.

ChartObj
- Arrow
- ChartCalendar
- CSV
- Dimension
- Point3D
- NearestPointData
- Polysurface
- ChartScale
  - LinearScale
  - LogScale
  - TimeScale
  - PointAndFigureScale
- UserCoordinates
  - WorldCoordinates
    - WorkingCoordinates
      - PhysicalCoordinates
        - CartesianCoordinates
          - PolarCoordinates
          - AntennaCoordinates
        - TimeCoordinates
        - PointAndFigureCoordinates
- ChartDataset
  - SimpleDataset
    - TimeSimpleDataset
    - ElapsedTimeSimpleDataset
    - ContourDataset
  - GroupDataset
    - TimeGroupDataset
    - ElapsedTimeGroupDataset
- AutoScale
  - LinearAutoScale
  - LogAutoScale
  - TimeAutoScale
  - ElapsedTimeAutoScale
  - PointAndFigureAutoScale
- MouseListener
  - MoveObj
  - FindObj
  - DataToolTip
  - ChartZoom
  - MagniView
  - MoveCoordinates

DataCursor
- MoveData

ChartAttribute

ChartGradient

ChartPrint

BufferedImage

System.Windows.Forms.UserControl

ChartView

Rectangle2D

Point2D

Point3D

GroupPoint2D

DoubleArray

DoubleArray2D

BoolArray

Polysurface

GraphObj
- AntennaAnnotation
- TickMark
- Axis
  - LinearAxis
    - PolarAxes
    - AntennaAxes
  - LogAxis
  - TimeAxis
  - PointAndFigureXAxis
  - PointAndFigureYAxis
- ChartText
  - ChartTitle
  - AxisTitle
  - ChartLabel
    - NumericLabel
      - BarDatapointValue
    - TimeLabel
    - ElapsedTimeLabel
    - StringLabel
  - AxisLabels
    - NumericAxisLabels
    - TimeAxisLabels
    - ElapsedTimeAxisLabels
    - StringAxisLabels
    - PolarAxesLabels
    - AntennaAxesLabels
    - PointAndFigureYAxisLabels
- 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
  - PolarPlot
    - PolarLinePlot
    - PolarScatterPlot
  - AntennaPlot
    - AntennaLinePlot

AntennaScatterPlot
AntennaLineMarkerPlot

Background
ChartImage

ChartShape
ChartSymbol
Marker
ChartZoom

5. Configuring QCTAChart Datasources

FinDataSourceBase
- **FinGenericHistoricalDataSource**
  - **FinGoogleHistoricalDataSource**
    - **FinGoogleURLHistoricalDataSource**
    - **FinGoogleCSVFileHistoricalDataSource**
    - **FinGoogleURLIntradayDataSource**
  - **FinMetaStockHistoricalDataSource**
    - **FinMetaStockCSVFileHistoricalDataSource**
    - **FinMetaStockURLHistoricalDataSource**
  - **FinQuandlHistoricalDataSource**
    - **FinQuandlCSVFileHistoricalDataSource**
    - **FinQuandlURLHistoricalDataSource**
  - **FinYahooHistoricalDataSource**
    - **FinYahooCSVFileHistoricalDataSource**
    - **FinYahooURLHistoricalDataSource**
    - **FinYahooURLIntradayDataSource**
- **FinGenericCurrentDataSource**
  - **FinQuandlCurrentDataSource**
    - **FinQuandlURLCurrentDataDataSource**
  - **FinYahooCurrentDataSource**
    - **FinYahooURLCurrentDataSource**

Before you can chart stocks, you need to acquire the historical OHLCV data for the stocks you want to consider. You might already have this information in files (one for each stock – usually in Metastock CSV 7- or 8-column format), or you might just want to grab data from a URL which supports the download of historical data (Yahoo, Google, or Quandl). Just keep in mind, while these data sources are free, you are not permitted to resell their data. So, you can write a program which acquires and displays the data, and resell that program. In that case, your customer would be accessing the data directly from the source (Yahoo, Google, and Quandl). You cannot introduce a charge, one time or recurring, to your customer for these free data feeds. You should familiarize yourself with the rules and regulations of these data feeds:

Yahoo - https://info.yahoo.com/guidelines/us/yahoo/ydn/ydn-3955.html

Google – since the Google Finance API has been deprecated, they have removed most all information about it from their site. Logically enough, you will find the remaining information by searching Google for "Google Finance API".

Quandl - http://www.quandl.com/about/terms

Metastock – Almost every historical stock data source sells data in the form of files using the Metastock ASCII CSV 7- or 8-column format.

These data sources support US, and non-US markets. Using this software you can customize the ticker symbol so that you can grab data from any source supported by the data source. That includes stock

markets in Europe, India, China and Japan. You will need to do some research to determine exactly what exchange symbol needs to be combined with the ticker symbol in order for it to work.

The most common format used in retrieving data from a URLs is the CSV (comma separated value) format. Saying that a URL uses a CSV format does not mean that all CSV formats are compatible. Each reader for a CSV data source must be custom configured for source, whether it is Yahoo, Google, or Quandl. Other formats include JSON and XML. Just like CSV data sources, the JSON and XML formats are not compatible across data sources; the JSON and XML data presentation from Yahoo is completely different that that of Quandl.. Our Yahoo and Quandl data readers for Yahoo, Google and Quandl are not going to be compatible with other data sources. There are bound to be strong similarities though, and we can be contracted to create a data reader for other data sources (CSV, XML, JSON or any other) using the same basic building blocks.

Getting Started with a Data Source

There are two types of data sources. The first is used to acquire historical stock data. The major sources of free historical data are Google, MetaStock, Quandl, and Yahoo. There are classes for reading data from a file, and others for reading direct from a URL.

Historical Data Source Classes

Google Historical Data Sources
- **FinGoogleURLHistoricalDataSource**
- **FinGoogleCSVFileHistoricalDataSource**
- **FinGoogleURLIntradayDataSource**

MetaStock Historical Data Source
- **FinMetaStockCSVFileHistoricalDataSource**

Quandl Historical Data Sources
- **FinQuandlCSVFileHistoricalDataSource**
- **FinQuandlURLHistoricalDataSource**

Yahoo Historical Data Sources
- **FinYahooCSVFileHistoricalDataSource**
- **FinYahooURLHistoricalDataSource**
- **FinYahooURLIntradayDataSource**

There are some differences between the data source classes, specifically when dealing with securities not found on the standard North American exchanges. If you can enter a simple stock ticker into the Yahoo quote engine, and have it find a specific stock, you can easily use the Yahoo historical data source. The same is true of Google. In these cases you are working with the stock ticker. There are exceptions though. While Yahoo is able to display on its own web sites historical data for some consolidated averages, the Dow 30 Industrial Average (Yahoo ticker symbol ^DJI), it will not permit external users to download the same data.

The first thing you need to do is to select a portfolio of stocks. For the purposes of this tutorial, we will acquire historical data for the following US stocks:

AAPL – Apple

INTC - Intel

IBM – IBM (International Business Machines)

TXN – Texas Instruments

AMAT – Applied Materials

CSCO – Cisco Systems

So you create a couple of string arrays.

C#

```
String[] idStrings = { "Intel", "IBM", "Tex Inst", "App Mat", "CSCO", "Apple", "QQQ" };
String[] tickerStrings = { "INTC", "IBM", "TXN", "AMAT", "CSCO", "AAPL", "QQQ" };
```

VB

```
Private  idStrings  As [String]() = { "Intel", "IBM", "Tex Inst", "App Mat", "CSCO",
"Apple", "QQQ"}
Private tickerStrings As [String]()={"INTC", "IBM", "TXN", "AMAT", "CSCO", "AAPL", "QQQ"}
```

The variable names are not important. The first array (idStrings) holds an array of strings which represent the displayed strings for the stocks. The second array holds the actual ticker look-up strings.

The *idStrings* can be anything that you want, though for practical reasons you should keep them short. If you want to use abbreviated versions of the actual stock name, you can do that. Or you can use the ticker strings. The *tickerStrings* ticker strings must be perfectly accurate, or else the stock look-up will fail.

URL-Based Data Sources

Yahoo URL-Based Data Source

To create a historical data source, instantiate one (FinYahooURLHistoricalDataSource below) using the default constructor. Add the stock names, and ticker symbols using the AddTickerLookupItem method.

C#

```
FinYahooURLHistoricalDataSource finStockHistoricalData = new
FinYahooURLHistoricalDataSource();

for (int i = 0; i < idStrings.Length; i++)
   finStockHistoricalData.AddTickerLookupItem(idStrings[i],tickerStrings[i]);
```

VB

```
Private finStockHistoricalData As New FinYahooURLHistoricalDataSource()

For i As Integer = 0 To idStrings.Length - 1
     finStockHistoricalData.AddTickerLookupItem(idStrings[i],tickerStrings[i])
Next
```

Google URL-Based Data Source

To create a historical data source, instantiate one (FinGoogleURLHistoricalDataSource below) using the default constructor. Add the stock names, and ticker symbols using the AddTickerLookupItem method.

C#

```
FinGoogleURLHistoricalDataSource finStockHistoricalData = new
FinGoogleURLHistoricalDataSource();
```

```
for (int i = 0; i < idStrings.Length; i++)
   finStockHistoricalData.AddTickerLookupItem(idStrings[i],tickerStrings[i]);
```

VB

```
Private finStockHistoricalData As New FinGoogleURLHistoricalDataSource()

For i As Integer = 0 To idStrings.Length - 1
     finStockHistoricalData.AddTickerLookupItem(idStrings(i), tickerStrings(i))
Next
```

Quandl URL-Based Data Source

If you use the Quandl (FinQuandlURLHistoricalDataSource) data source, you may need to specify a Quandl data source folder when calling the constructor. All North American stocks are found in the Quandl WIKI folder. So the FinQuandlURLHistoricalDataSource code looks like.

C#

```
String[] idStrings ={"Apple", "Intel", "IBM","Tex. Inst.","App. Mat.", "Cisco"};
String[] tickerStrings ={ "AAPL", "INTC", "IBM", "TXN", "AMAT", "CSCO"};

FinQuandlURLHistoricalDataSource finStockHistoricalData =
    new FinQuandlURLHistoricalDataSource("WIKI");

for (int i = 0; i < idStrings.Length; i++)
    finStockHistoricalData.AddTickerLookupItem(idStrings[i],tickerStrings[i]);
```

VB

```
Dim idStrings As [String]() = {"Apple","Intel","IBM","Tex. Inst.","App. Mat.","Cisco"}
Dim tickerStrings As [String]() = {"AAPL","INTC","IBM","TXN","AMAT","CSCO"}

Dim finStockHistoricalData As New FinQuandlURLHistoricalDataSource("WIKI")

For i As Integer = 0 To idStrings.Length - 1
  finStockHistoricalData.AddTickerLookupItem(idStrings(i), tickerStrings(i))
```

C#

```
FinQuandlURLHistoricalDataSource finStockHistoricalData =
    new FinQuandlURLHistoricalDataSource();
```

VB

```
Dim finStockHistoricalData As New FinQuandlURLHistoricalDataSource()
```

Special Note – You need to get a Quandl token from the Quandl web site (https://www.quandl.com/) if you plan to access the Quandl databases more than a fifty times a day. The token can be obtained from Quandl and it is free. You will have to provide some simple registration information to the Quandl web site in return for the token. Once you have received the token, you set it in the software using the FinChartConstants.QuandlToken static property.

Sign up here: https://www.quandl.com/users/sign_up

Once you sign up, you can view your unique token here: https://www.quandl.com/help/api

If that link doesn't work, look for Auth Token on this Quandl web page: https://www.quandl.com/help/api and follow the directions. You must be signed up with Quandl though.

C#
```
FinChartConstants.QuandlToken="9YVaMHshys5vKnqr4kak"; // SIMILAR TO BUT NOT A VALID TOKEN
```

VB
```
FinChartConstants.QuandlToken="9YVaMHshys5vKnqr4kak" ' SIMILAR TO BUT NOT A VALID TOKEN
```

If you specify WIKI in the constructor, the software assumes that all ticker symbols you enter are found in the Quandl WIKI folder. WIKI is also the default value, so if you use the empty constructor for FinQunadlURLHistoricalDataSource, it is the same as specifying "WIKI".

This example defaults to the “WIKI” data source folder.

C#
```
FinQuandlURLHistoricalDataSource finStockHistoricalData =
    new FinQuandlURLHistoricalDataSource();
```

VB
```
Dim finStockHistoricalData As New FinQuandlURLHistoricalDataSource()
```

WIKI is not the only Quandl data source supported. The software also supports non-North American stocks. You can specify the following non-US markets:

NSE	National Stock Exchange India
BSE	Bombay Stock Exchange
FSE	Frankfurt Stock Exchange
HKEX	Hong Kong Stock Exchange
LSE	London Stock Exchange
SSE	Boerse Stuttgart Stock Exchange
TSE	Tokyo Stock Exchange

If you specify one of these other data sources, you MUST get the ticker name exactly right, or else the stock data will fail to load. Ticker names are found on the Quandl web site: http://www.quandl.com/help/api-for-stock-data, under the heading *Stock Price Data Source*. Click on the links for a detailed description of what stocks are available, and their ticker symbols.

For example, if you wanted to access data on the NSE (India) exchange, you could do so using the following initialization:

C#

```
String[] idStrings =
    {"NIFTY Index","Oil India","Spanco","Northgate","MVL Ind.", "Zee News" };
String[] tickerStrings ={"SPCNXNIFTY ","OIL","SPANCO","NORTHGATE", "MVLIND", "ZEENEWS"};

FinQuandlURLHistoricalDataSource finStockHistoricalData =
    new FinQuandlURLHistoricalDataSource("NSE");

for (int i = 0; i < idStrings.Length; i++)
     finStockHistoricalData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
```

VB

```
Dim idStrings As [String]() =
    {"NIFTY Index","Oil India","Spanco", "Northgate","MVL Ind.","Zee News"}
Dim tickerStrings As [String]() =
     {"SPCNXNIFTY","OIL","SPANCO","NORTHGATE","MVLIND","ZEENEWS"}
Dim finStockHistoricalData As New FinQuandlURLHistoricalDataSource(NSE)

For i As Integer = 0 To idStrings.Length - 1
 finStockHistoricalData.AddTickerLookupItem(idStrings(i), tickerStrings(i))
```

There is one issue which makes importing data from the Quandl data sources complicated, though it is a complication we take care of automatically. The Quandl exchanges all use unique column formats for their data. Not only that, most of the exchanges do not even use a standardized column format for the data within the given exchange. So what our software does is read the column headings for each downloaded stock, and assign the resulting column of data to the associated element of an OHLCV (Open-High-Low-Close-Volume) element in our own own software. This is further complicated by the fact that the exchanges all use different wording for the same column heads. For example, for the daily Volume value, you find column headings of: "Volume", "Adj. Volume", "Share Volume", "SharesTraded", "Total Trade Quantity", "No. of Shares", "Traded Volume". We tried to seek out every variant and take it into account. A complete list of the different column headings we found are listed below. We didn't check every stock of every exchange and may have missed one. If so, let us know.

OHLCV	**Stock Data Column Head**
Date	Date
Open	Open, Adj. Open, Previous Close, Last Close, Previous Day Price
High	High, Adj. High
Low	Low, Adj. Low
Close	Close, Adj. Close, Last Traded, Nominal Price, Price
Volume	Volume, Adj. Volume, Share Volume, SharesTraded, Total Trade Quantity, No. of Shares, Traded Volume

If the stock data has both a non-adjusted column (Close for example) and the equivalent adjusted column (Adj. Close for example), the adjusted column is used, to better take into account stock splits. In some cases (Open and Last Close for example) the equivalence is not exact. Never less, the column headings returned for a given stock are deemed important for that exchange, and we map them as depicted in the table above.

The FinYahooURLHistoricalDataSource and FinQuandlURLHistoricalDataSource data source classes also include support for reading data in XML and JSON formats. The underlying structures of the Yahoo and Quandl XML and JSON file formats are too complicated and verbose to list out. But reading them is simple. Just set the DataFormat property of the Yahoo or Quandl data source object to the desired data format.

Important Note: Yahoo limits the download of historical data using JSON and XML protocols to one year of data. If you request more than one year, the download will simply fail. Right now, if you request more than one year of data when using JSON or XML, the request is truncated to one year. Should this Yahoo-based limitation change, we will change the truncation to the newer values.

C#

```
FinQuandlURLHistoricalDataSource finStockHistoricalData =
    new FinQuandlURLHistoricalDataSource();
finStockHistoricalData.DataFormat = FinChartConstants.JSON_FORMAT;
```

VB

```
Dim finStockHistoricalData As New FinQuandlURLHistoricalDataSource()
finStockHistoricalData.DataFormat = FinChartConstants.JSON_FORMAT
```

or

C#

```
finStockHistoricalData.DataFormat = FinChartConstants.XML_FORMAT;
```

VB

```
finStockHistoricalData.DataFormat = FinChartConstants.XML_FORMAT
```

The default is DataFormat = FinChartConstants.CSV_FORMAT.

Local File-Based Data Sources

The file-based data sources include Metastock, Yahoo and Quandl. They are instantiated much the same as the URL-based data sources, with the following differences. In the constructor, you specify a source directory (folder) as part of the initialization. And the ticker strings are the filenames of the stock data files in the specified folder.

Metastock File-Based Data Source

If you have data in local file acquired from a stock service, most likely it is in MetaStock ASCII (non-binary) format. You can initialize the FinMetaStockCSVFileHistoricalDataSource using code similar to that below. The source directory (folder) is specified as a parameter in the FinMetaStockCSVFileHistoricalDataSource constructor.

C#

```
String[] idStrings = {"Apple","Intel","IBM", "Tex. Inst.", "App. Mat.", "Cisco"};
String[] stockFilenames = {"AAPL","INTC", "IBM", "TXN", "AMAT", "CSCO" };

// Absolute folder example
// String folder = @"c:\Quinn-Curtis\DotNet\QCTAChart\DataFiles\MetastockData";

// Relative folder example
String  folder = @"..\..\..\..\..\..\DataFiles\MetastockData";


FinMetaStockCSVFileHistoricalDataSource finStockHistoricalData =
    new FinMetaStockCSVFileHistoricalDataSource(folder);

for (int i = 0; i < idStrings.Length; i++)
    finStockHistoricalData.AddTickerLookupItem(idStrings[i], stockFilenames[i]);
```

Important Note: In C#, you can get rid of the double backslashes in the folder specification by preceding the string with the @ symbol. That is a C# thing and does not apply to VB.

For example:

```
String  folder = @"..\..\..\..\..\..\DataFiles\MetastockData";
```

VB

```
Dim idStrings As [String]() = {"Apple","Intel","IBM","Tex. Inst.","App. Mat.","Cisco"}
Dim stockFilenames As [String]() = {"AAPL","INTC","IBM","TXN","AMAT","CSCO"}

Dim folder As [String] ="c:\Quinn-Curtis\DotNet\QCTAChart\DataFiles\MetastockData"

Dim finStockHistoricalData As New FinMetaStockCSVFileHistoricalDataSource(folder)
```

```
For i As Integer = 0 To idStrings.Length - 1
 finStockHistoricalData.AddTickerLookupItem(idStrings(i), stockFilenames(i))
Next
```

In this case, the historicalTickerLookup specifies the filenames of the files in the specified folder. The filenames are specified without an extension. The extension is assumed to be CSV. So, if your stock data file uses a different extension, you should make a copy of it and change the copy's extention to CSV. In order for the example above to work, you need to have valid historical stock data files in the folder "C:\Quinn-Curtis\MetastockData".

```
C:
     \Quinn-Curtis\DotNet\QCTAChart\MetastockData
        AAPL.CSV
        INTC.CSV
        IBM.CSV
        TXN.CSV
        AMAT.CSV
        CSCO.CSV
```

MetaStock data files ASCII data files are usually in either an 7-column format, or a 8-column format. They usually include a line of column headings. A typical 7-column formatted file looks like:

<ticker>,	<date>,	<open>,	<high>,	low>,	<close>,	<vol>
INTC,	20140110,	25.5,	25.85,	25.5,	25.53,	30620900
INTC,	20140113,	25.6,	25.99,	25.42,	25.5,	40304800
INTC,	20140114,	26,	26.55,	25.9,	26.51,	74158100
INTC,	20140115,	26.72,	27.12,	26.61,	26.67,	58315200
INTC,	20140116,	26.64,	26.74,	26.33,	26.54,	49521700
INTC,	20140117,	25.36,	25.86,	25.25,	25.85,	112538000

While we have inserted tabs to make the table more readable, the source data file will not include tabs at the comma breaks.

The 8-column format adds an <interval> column, in this case specifying that the interval is daily.

<ticker>,	<period>,	<date>,	<open>,	<high>,	<low>,	<close>,	<vol>
INTC,	D,	20140110,	25.5,	25.85,	25.5,	25.53,	30620900
INTC,	D,	20140113,	25.6,	25.99,	25.42,	25.5,	40304800
INTC,	D,	20140114,	26,	26.55,	25.9,	26.51,	74158100
INTC,	D,	20140115,	26.72,	27.12,	26.61,	26.67,	58315200
INTC,	D,	20140116,	26.64,	26.74,	26.33,	26.54,	49521700
INTC,	D,	20140117,	25.36,	25.86,	25.25,	25.85,	112538000

In the example above the data is still EOD (End of Day) data. If the data consisted of intraday data, then each item would include an extended <date>, which incorporated the time of day. There are several variants of the date/time fields, and include:"<DATE>", "<DTYYYYMMDD>", "<DTYYMMDD>", and "<TIME>".

If the example above was intra-day data, then the file would need to include a time-of-day column. It looks like most providers simply drop the period column. So the file looks like:

<ticker>,	<date>,	<time>	<open>,	<high>,	<low>,	<close>,	<vol>
INTC,	20140110,	1250,	31.40,	31.41,	31.38,	31.00	3991974
INTC,	20140110,	1245,	31.41,	31.42,	31.40,	31.41,	100552
INTC,	20140110,	1240,	31.42,	31.44,	31.41,	31.41,	85763
INTC,	20140110,	1235,	31.42,	31.44,	31.41,	31.42,	216606
INTC,	20140110,	1230,	31.42,	31.44,	31.42,	31.42,	101789

We have also seen instances where the time-of-day value is appended directly to to the data value, without adding a new column.

<ticker>,	<date>,	<open>,	<high>,	<low>,	<close>,	<vol>
INTC,	201401101250,	31.40,	31.41,	31.38,	31.00	3991974
INTC,	201401101245,	31.41,	31.42,	31.40,	31.41,	100552
INTC,	201401101240,	31.42,	31.44,	31.41,	31.41,	85763
INTC,	201401101235,	31.42,	31.44,	31.41,	31.42,	216606
INTC,	201401101230,	31.42,	31.44,	31.42,	31.42,	101789

Because they have a unique string length, the software can automatically take into account the following MetaStock <date> formats: yyMMdd, yyyyMMdd and yyyyMMddHHmm. If a <time> field is present, the software can also differentiate between HHmm and HHmmss time formats.

Assuming that the column headings are used correctly, our software should be able to identify the format and load the data appropriately. The actual MetaStock format definition does not require that the columns appear in the order above. The software takes that into account too.

Not all MetaStock data sources include the header row though. In that case the software defaults to the MetaStock 7-column format described above.

Yahoo, Google and Quandl also permit the download of EOD (End-of day) data in CSV format. Unfortunately the formats are similar to, yet different from, the MetaStock format. So in if you have data downloaded from one of those providers, you need to use the custom data source classes we created specifically for them.

while the Google format looks like:

Date,Open,High,Low,Close,Volume

11-Jul-14,31.26,31.45,31.04,31.25,20053217

10-Jul-14,30.60,31.33,30.44,31.26,32458765

9-Jul-14,30.84,30.98,30.69,30.89,28302427

8-Jul-14,31.00,31.08,30.70,30.79,37614506

7-Jul-14,31.09,31.20,30.92,31.03,22237033

3-Jul-14,31.08,31.36,31.02,31.14,20437633

2-Jul-14,30.99,31.05,30.80,30.98,16831171

Note that they use different Date formats, and that the Yahoo data files also includes an adjusted close column after the Volume column. The Google date format uses a .Net d-MMM-yy format, while the Yahoo date format uses a .Net yyyy-MM-dd format.

Yahoo File-Based Data Source

If you download historical data directly from Yahoo, and save it to disk, the CSV format is the same as accessing directly from the URL. But, since you want to access it from a data file, you initialize things similar to the way described in the Metastock example. You can initialize the FinYahooCSVFileHistoricalDataSource using code similar to that below. The source directory (folder) is specified as a parameter in the FinYahooCSVFileHistoricalDataSource constructor.

C#

```
FinChartData finChartData = null;

ChartCalendar startDate = new ChartCalendar();
ChartCalendar stopDate = new ChartCalendar();

String[] idStrings =  {  "INTC",  "IBM","TXN","AMAT","CSCO","AAPL","QQQ"};

String[] stockFilenames ={"INTC","IBM","TXN","AMAT","CSCO","AAPL","QQQ"};
FinYahooCSVFileHistoricalDataSource finStockHistoricalData = null;

stopDate.Add(ChartObj.DAY_OF_YEAR,-1);
startDate.Add(ChartObj.YEAR,-15);

// Absolute path
// String datafolder = @"C:\Quinn-Curtis\DotNet\QCTAChart\DataFiles\Yahoo";
// Relative path
String datafolder = @"..\..\..\..\..\..\DataFiles\Yahoo";

finStockHistoricalData = new FinYahooCSVFileHistoricalDataSource(datafolder);

for (int i = 0; i < stockFilenames.Length; i++)
{
  finStockHistoricalData.AddTickerLookupItem(idStrings[i], stockFilenames[i]);
}

finChartData =
    new FinChartData (finStockHistoricalData, stockFilenames, startDate, stopDate);

InitFinChartView(finChartData);
```

VB

```
Private finChartData As FinChartData = Nothing

Private startDate As New ChartCalendar()
Private stopDate As New ChartCalendar()

Private idStrings As [String]() = {"INTC", "IBM", "TXN", "AMAT", "CSCO", "AAPL", "QQQ"}
```

```
Private stockFilenames As [String]() = {"INTC", "IBM", "TXN", "AMAT", "CSCO", "AAPL",
"QQQ"}
Private finStockHistoricalData As FinYahooCSVFileHistoricalDataSource = Nothing

stopDate.Add(ChartObj.DAY_OF_YEAR, -1)
startDate.Add(ChartObj.YEAR, -8)

' Absolute path
'   Dim datafolder As [String] = "C:\Quinn-Curtis\DotNet\QCTAChart\DataFiles\Yahoo"
' Relative path
Dim datafolder As [String] = "..\..\..\..\..\..\DataFiles\Yahoo"

finStockHistoricalData = New FinYahooCSVFileHistoricalDataSource(datafolder)

For i As Integer = 0 To stockFilenames.Length - 1
    finStockHistoricalData.AddTickerLookupItem(idStrings(i), stockFilenames(i))
Next

finChartData =
    New FinChartData(finStockHistoricalData, stockFilenames, startDate, stopDate)

InitFinChartView(finChartData)
```

The Yahoo format looks like:

```
Date,Open,High,Low,Close,Volume,Adj Close
2014-12-18,36.75,37.02,36.43,37.02,31086000,37.02
2014-12-17,35.62,36.33,35.33,36.24,31637700,36.24
2014-12-16,35.86,36.50,35.56,35.56,31166300,35.56
2014-12-15,36.39,36.78,35.90,35.92,30633400,35.92
2014-12-12,36.45,36.82,36.22,36.23,29322500,36.23
2014-12-11,36.50,37.21,36.44,36.70,25088200,36.70
2014-12-10,36.93,37.02,36.30,36.42,27499900,36.42
2014-12-09,36.67,37.09,36.30,36.89,28487300,36.89
2014-12-08,37.46,37.73,36.94,37.20,28134800,37.20
2014-12-05,37.57,37.90,37.52,37.67,20527600,37.67
2014-12-04,37.40,37.46,37.04,37.46,23359100,37.46
2014-12-03,37.68,37.89,37.41,37.43,30660800,37.43
```

Special Note on Adjusted Values

If our software sees an Adjusted Close column, it checks to see if the Adjusted Close value is different than the Close values. If so, it assumes that Adjusted Close value is the Close value normalized for stock splits. It then re-calculates adjusted values for the Open, High, Low and Volume (using Adjusted Close/Close), so that they represent the same normalization as the Adjusted Close. The adjusted values are then plotted in the charts. This keeps stock splits from introducing discountinuties in the OHLC data on the date of the stock split.

Quandl File-Based Data Source

If you download historical data directly from Quandl, and save it to disk, the CSV format is the same as accessing directly from the URL. But, since you want to access it from a data file, you initialize things similar to the way described in the Metastock and Yahoo examples. You can initialize the FinQuandlCSVFileHistoricalDataSource using code similar to that below. The source directory (folder) is specified as a parameter in the FinQuandlCSVFileHistoricalDataSource constructor. The default file names used by Quandl are the ticker symbol prefaced by the Quandl data source, WIKI in the example below.

C#

```
FinChartData finChartData = null;

ChartCalendar startDate = new ChartCalendar();
ChartCalendar stopDate = new ChartCalendar();

String[] idStrings = { "AAPL", "INTC", "IBM", "TXN", "AMAT", "CSCO" };
// In the case of files, the ticker lookup specifies the file name, not the ticker
symbol, file extension csv is assumed.
String[] stockFilenames = { "WIKI-AAPL", "WIKI-INTC", "WIKI-IBM", "WIKI-TXN", "WIKI-
AMAT", "WIKI-CSCO" };

 // Since this is a file, it will truncate the file data to these dates
stopDate.Add(ChartObj.DAY_OF_YEAR, -1);
startDate.Add(ChartObj.YEAR, -8);

// Absolute path
 //   String datafolder = @"C:\Quinn-Curtis\DotNet\QCTAChart\DataFiles\Quandl WIKI";
// Relative path
String datafolder = @"..\..\..\..\..\..\..\DataFiles\Quandl WIKI";
FinQuandlCSVFileHistoricalDataSource finStockHistoricalData = new
FinQuandlCSVFileHistoricalDataSource(datafolder);

for (int i = 0; i < idStrings.Length; i++)
{
    finStockHistoricalData.AddTickerLookupItem(idStrings[i], stockFilenames[i]);
}
finChartData = new FinChartData(finStockHistoricalData, idStrings, startDate, stopDate);

InitFinChartView(finChartData);
```

VB

```
Private finChartData As FinChartData = Nothing

Private startDate As New ChartCalendar()
Private stopDate As New ChartCalendar()

Private idStrings As [String]() = {"AAPL", "INTC", "IBM", "TXN", "AMAT", "CSCO"}
' In the case of files, the ticker lookup specifies the file name, not the ticker symbol,
file extension csv is assumed.
Private stockFilenames As [String]() = {"WIKI-AAPL", "WIKI-INTC", "WIKI-IBM", "WIKI-TXN",
"WIKI-AMAT", "WIKI-CSCO"}

stopDate.Add(ChartObj.DAY_OF_YEAR, -1)
startDate.Add(ChartObj.YEAR, -8)

' Absolute path
'   Dim datafolder As [String] = "C:\Quinn-Curtis\DotNet\QCTAChart\DataFiles\Quandl WIKI"
' Relative path
Dim datafolder As [String] = "..\..\..\..\..\..\..\DataFiles\Quandl WIKI"
Dim finStockHistoricalData As New FinQuandlCSVFileHistoricalDataSource(datafolder)

For i As Integer = 0 To idStrings.Length - 1
    finStockHistoricalData.AddTickerLookupItem(idStrings(i), stockFilenames(i))
Next

finChartData = New FinChartData(finStockHistoricalData, idStrings, startDate, stopDate)

InitFinChartView(finChartData)
```

The Quandl WIKI format looks like:

Date,Open,High,Low,Close,Volume,Ex-Dividend,Split Ratio,Adj. Open,Adj. High,Adj. Low,Adj. Close,Adj. Volume
2014-12-18,36.78,37.02,36.43,37.02,31301981.0,0.0,1.0,36.78,37.02,36.43,37.02,31301981.0
2014-12-17,35.62,36.33,35.33,36.24,31043291.0,0.0,1.0,35.62,36.33,35.33,36.24,31043291.0
2014-12-16,35.82,36.5,35.56,35.56,30868335.0,0.0,1.0,35.82,36.5,35.56,35.56,30868335.0
2014-12-15,36.39,36.783,35.9,35.92,30292187.0,0.0,1.0,36.39,36.783,35.9,35.92,30292187.0

2014-12-11,36.44,37.21,36.44,36.7,24821993.0,0.0,1.0,36.44,37.21,36.44,36.7,24821993.0
2014-12-10,36.93,37.02,36.3,36.42,27257663.0,0.0,1.0,36.93,37.02,36.3,36.42,27257663.0
2014-12-09,36.68,37.09,36.3,36.89,28460487.0,0.0,1.0,36.68,37.09,36.3,36.89,28460487.0
2014-12-05,37.52,37.9,37.52,37.67,19019870.0,0.0,1.0,37.52,37.9,37.52,37.67,19019870.0
2014-12-04,37.39,37.46,37.04,37.46,23391744.0,0.0,1.0,37.39,37.46,37.04,37.46,23391744.0
2014-12-02,37.26,37.6,37.18,37.6,28352943.0,0.0,1.0,37.26,37.6,37.18,37.6,28352943.0
2014-12-01,37.23,37.62,36.9,37.17,30715970.0,0.0,1.0,37.23,37.62,36.9,37.17,30715970.0
2014-11-28,37.0,37.69,36.94,37.25,19128510.0,0.0,1.0,37.0,37.69,36.94,37.25,19128510.0
2014-11-26,36.31,36.99,36.28,36.9,24054283.0,0.0,1.0,36.31,36.99,36.28,36.9,24054283.0

Since the Quandl data includes Adjusted Open, Adjusted High, Adjusted Low, Adjusted Close and Adjusted Volume, those values are used, rather than the unadjusted open-high-low-close-volume data.

Google File-Based Data Source

If you download historical data directly from Google, and save it to disk, the CSV format is the same as accessing directly from the URL. But, since you want to access it from a data file, you initialize things similar to the way described in the Metastock example. You can initialize the FinGoogleCSVFileHistoricalDataSource using code similar to that below. The source directory (folder) is specified as a parameter in the FinGoogleCSVFileHistoricalDataSource constructor.

C#

```
FinChartData finChartData = null;

ChartCalendar startDate = new ChartCalendar();
ChartCalendar stopDate = new ChartCalendar();

String[] idStrings ={"INTC","IBM","TXN","AMAT","CSCO","AAPL","QQQ"};

String[] stockFilenames ={"INTC","IBM","TXN","AMAT","CSCO","AAPL","QQQ"};
FinGoogleCSVFileHistoricalDataSource finStockHistoricalData = null;

stopDate.Add(ChartObj.DAY_OF_YEAR,-1);
startDate.Add(ChartObj.YEAR,-15);

// Absolute path
// String datafolder = @"C:\Quinn-Curtis\DotNet\QCTAChart\DataFiles\Google";
// Relative path
String datafolder = @"..\..\..\..\..\..\DataFiles\Google";

finStockHistoricalData =FinGoogleCSVFileHistoricalDataSource(datafolder);

for (int i = 0; i < stockFilenames.Length; i++)
{
  finStockHistoricalData.AddTickerLookupItem(idStrings[i], stockFilenames[i]);
}

finChartData =
    new FinChartData (finStockHistoricalData, stockFilenames, startDate, stopDate);

InitFinChartView(finChartData);
```

VB

```
Private finChartData As FinChartData = Nothing

Private startDate As New ChartCalendar()
Private stopDate As New ChartCalendar()
```

```
Private idStrings As [String]() = {"INTC", "IBM", "TXN", "AMAT", "CSCO", "AAPL", "QQQ"}

Private stockFilenames As [String]() = {"INTC", "IBM", "TXN", "AMAT", "CSCO", "AAPL",
"QQQ"}
Private finStockHistoricalData As FinGoogleCSVFileHistoricalDataSource = Nothing

stopDate.Add(ChartObj.DAY_OF_YEAR, -1)
startDate.Add(ChartObj.YEAR, -8)

' Absolute path
'   Dim datafolder as [String] = "C:\Quinn-Curtis\DotNet\QCTAChart\DataFiles\Google"
' Relative path
Dim datafolder As [String] = "..\..\..\..\..\..\DataFiles\Google"

finStockHistoricalData = New FinGoogleCSVFileHistoricalDataSource(datafolder)

For i As Integer = 0 To stockFilenames.Length - 1
    finStockHistoricalData.AddTickerLookupItem(idStrings(i), stockFilenames(i))
Next

finChartData =
    New FinChartData(finStockHistoricalData, stockFilenames, startDate, stopDate)

InitFinChartView(finChartData)
```

The Google format looks like:

Date,Open,High,Low,Close,Volume
22-Dec-14,36.39,37.26,36.39,37.21,29227071
19-Dec-14,37.02,37.16,36.23,36.37,76786814
18-Dec-14,36.75,37.02,36.43,37.02,32128124
17-Dec-14,35.62,36.33,35.33,36.24,31637748
16-Dec-14,35.86,36.50,35.56,35.56,31166334
15-Dec-14,36.39,36.78,35.90,35.92,30633423
12-Dec-14,36.45,36.82,36.22,36.22,29322529
11-Dec-14,36.50,37.21,36.44,36.70,25090084
10-Dec-14,36.93,37.02,36.30,36.42,27499869
9-Dec-14,36.67,37.09,36.30,36.89,28487318
8-Dec-14,37.46,37.73,36.94,37.20,28134819
5-Dec-14,37.57,37.90,37.52,37.67,20527600
4-Dec-14,37.40,37.46,37.04,37.46,23397863
3-Dec-14,37.68,37.89,37.41,37.43,30660821
2-Dec-14,37.18,37.60,37.18,37.60,28361104
1-Dec-14,37.21,37.62,36.90,37.17,30904004
28-Nov-14,37.04,37.69,36.94,37.25,19128510
26-Nov-14,36.37,36.99,36.28,36.90,24062025

Unlike the Yahoo and Quandl data, the Google data does not include any adjusted value columns. This is because all Google data items represent adjusted values in the raw download.

How to acquire the historical data values used in the charts.

You may want to acquire the historical OHLCV data values used in the creation of the charts. The values are stored internally in the FinChartData data structures. A typical initialization of the FinChartData class is shown below.

```
FinChartData finChartData = null;

ChartCalendar startDate = new ChartCalendar();
ChartCalendar stopDate = new ChartCalendar();
```

```
String[] idStrings = { "Intel", "IBM", "Tex Inst", "App Mat", "CSCO", "Apple", "QQQ" };
String[] tickerStrings = { "INTC", "IBM", "TXN", "AMAT", "CSCO", "AAPL", "QQQ" };

FinYahooURLCurrentDataSource finStockData = null;
FinYahooURLHistoricalDataSource finStockHistoricalData = null;

...

stopDate.Add(ChartObj.DAY_OF_YEAR, -1);
startDate.Add(ChartObj.YEAR, -10);
finStockData = new FinYahooURLCurrentDataSource();
finStockHistoricalData = new FinYahooURLHistoricalDataSource();

for (int i = 0; i < idStrings.Length; i++)
{
    finStockHistoricalData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
    finStockData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
}

finChartData =
   new FinChartData(finStockHistoricalData, finStockData, idStrings, startDate, stopDate);
```

Assuming that the primary chart has already been displayed, an indication that the data has already been read from the URL or file, you can retrieve the data using code similar to below.

Retrieve the data using the stocks ID, as specified in the idString array above, "Intel" in this case.

```
FinTimeSeriesDataset ds = this.ChartData.GetTimeSeriesDatsetById("Intel");
```

Retrieve the data using the stocks index, since Intel was the first item in the idStrings array above, it has an index of 0.

```
FinTimeSeriesDataset ds = this.ChartData.GetTimeSeriesDatsetByIndex(0);
```

Retrieve the data using the stocks ticker symbol, as specified in the tickerStrings array above.

```
FinTimeSeriesDataset ds = this.ChartData.GetTimeSeriesDatsetByTicker("INTC");
```

The FinTimeSeriesData is a subclass of the EventGroupDataset class, and can be used with all of the group plotting routines found in the QCChart2D software package. For OHLCV data, the plot types you will most likely be using are OHLCPlot and CandlestickPlot. You would reference the FinTimeSeriesDataset (*ds* in the example below) in the OHLC or CandlestickPlot constructor, as in the example below.

```
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, ds, 0.5, defaultattrib, fillattrib);
thePlot1.SetFastClipMode(ChartObj.FASTCLIP_X);
chartVu.AddChartObject(thePlot1);
```

The **FinChartData.GetTimeSeriesDataset...** routines assume that the data has already been acquired, i.e. they do not trigger a call to the underlying URL, or file, to read the data. If you are not displaying the data in a chart, you will probably need to force a read of the data, using the FinChartData.GetFinChartData() method. For example, for a read of the URL data, and then retrieve the data as a FinTimeSeriesData object.

```
finChartData.GetFinChartData();
FinTimeSeriesDataset ds = this.ChartData.GetTimeSeriesDatsetById("Intel");
```

The FinTimeSeriesDataset class is a subclass of the EventGroupDataset class. That is because for each x-value (a date/time value) in the group data set, there are multiple y-values (Open-high-low-close-volume) As such, it cannot be directly input to the plotting clases which expect a data set derived from a SimpleDataset. Plotting classes in the simple category are the SimpleLinePlot, SimpleLineMarkerPlot, SimpleBarPlot and SimpleScatterPlot classes.

These classes expect a dataset which has a single y-value for each x-value. But it may be that you want to just plot a simple line plot of close values for a stock. Or you might need to plot a bar plot of the volume values for a stock. In that case, you need to convert the source EventGroupDataset to an EventSimpleDataset. One of the methods in the EventGroupDataset class is a conversion utility for doing that. It takes one argument, shich is the index of which value you want to convert (Open = 0, High = 1, Low = 2, Close = 3, and Volume = 4). The example below creates a EventSimpleDataset of Close (index # 3) values for the stock Intel.

```
finChartData.GetFinChartData();
FinTimeSeriesDataset ds = this.ChartData.GetTimeSeriesDatsetById("Intel");

EventSimpleDataset esds = ds.ConvertToEventSimpleDataset(3);
```

You can then reference the *esds* dataset in the creation of the constructors of the SimpleLinePlot, SimpleLineMarkerPlot, SimpleBarPlot and SimpleScatterPlot .

```
 ChartAttribute attrib2 = new ChartAttribute(Color.Blue, 1, 0);
 SimpleLinePlot thePlot2 = new SimpleLinePlot(pTransform1, esds, attrib2);
 thePlot2.SetFastClipMode(ChartObj.FASTCLIP_X);
 chartVu.AddChartObject(thePlot2);
```

The descriptions above describe how you can retrieve historical stock data from the FinChartData class used to supply the charts with data. You can also retrieve historical data directly from the historical data source class you use. In the example above, this was the FinYahooURLHistoricalDataSource, declared and initited using code similar to below.

```
String[] idStrings = { "Intel", "IBM", "Tex Inst", "App Mat", "CSCO", "Apple", "QQQ" };
String[] tickerStrings = { "INTC", "IBM", "TXN", "AMAT", "CSCO", "AAPL", "QQQ" };


FinYahooURLHistoricalDataSource finStockHistoricalData = null;

...

finStockHistoricalData = new FinYahooURLHistoricalDataSource();

for (int i = 0; i < idStrings.Length; i++)
{
    finStockHistoricalData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
}
```

At this point, you can retrieve historical data directly from the historical data source, using the GetHistoricalData method call. In this case you pass in the stock ID, a start and ending data, and a frequency indicator ("d" for daily, "w" for weekly, "m").

```
ChartCalendar startDate = new ChartCalendar();
ChartCalendar stopDate = new ChartCalendar();

stopDate.Add(ChartObj.DAY_OF_YEAR, -1); // yesterday
startDate.Add(ChartObj.YEAR, -10); // 10 years ago

EventGroupDataset egds =
    finStockHistoricalData.GetHistoricalData("Intel", startDate, stopDate, "d");

EventSimpleDataset esds =
    egds.ConvertToEventSimpleDataset(3); // third index are close values
```

The quote frequency parameter only applies to the FinYahooURLHistoricalDataSource. The other URL data source only return EOD (End-of-day) data.

Current Financial Data

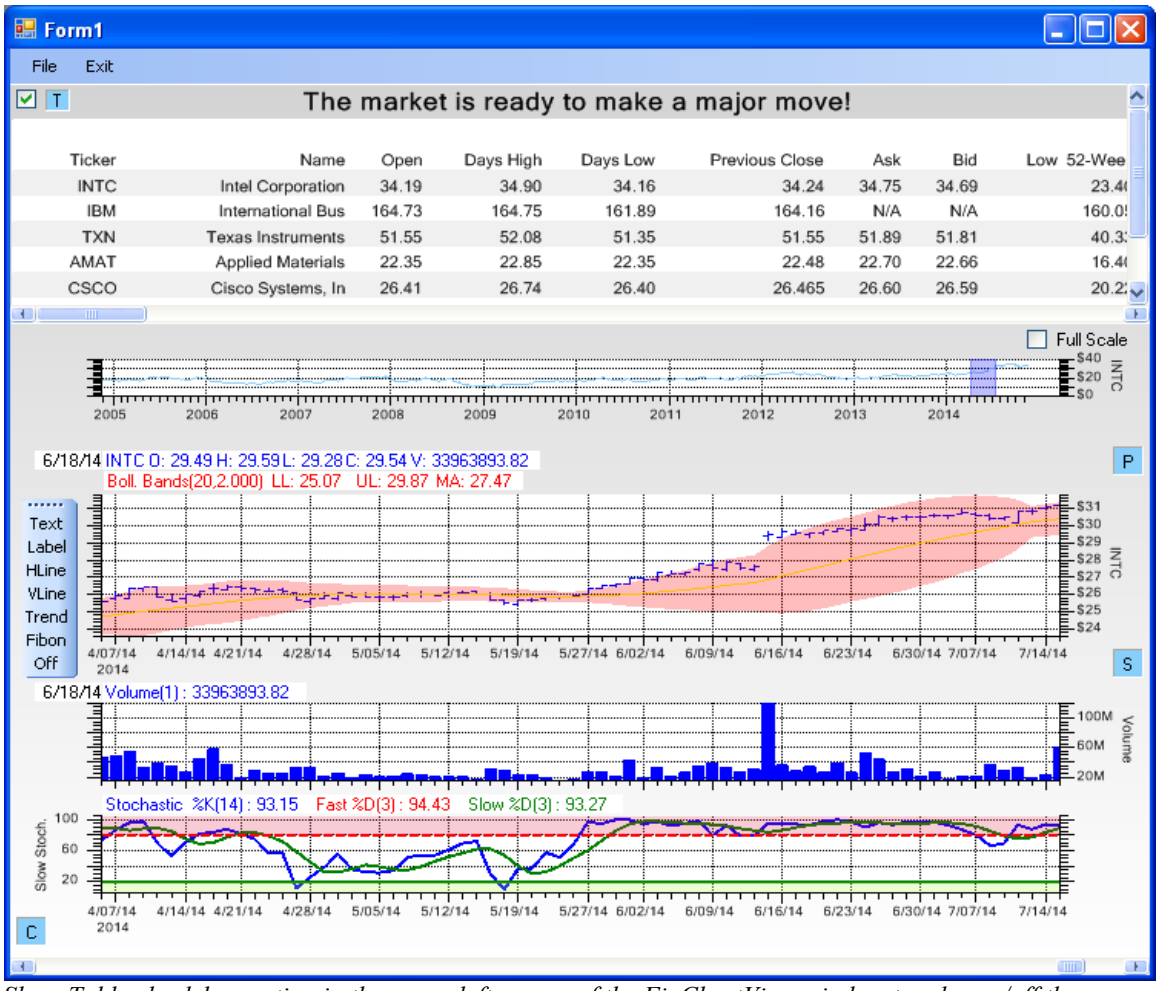
The previous discussion has been about historical data. There is another type of data which the software will display, and that is current financial data. Current financial data is not charted, but is instead optionally displayed in a table at the top of the FinChartView window. This data is available by URL only, and only from the Yahoo and Quandl data sources. It contains only stocks found on the US exchanges; it can't be used for other exchanges.

Quandl Current Data Source

FinQuandlURLCurrentDataDataSource

Yahoo Current Data Source

FinYahooURLCurrentDataSource



The Show Table check box option in the upper left corner of the FinChartView window toggles on/off the current financial data table.

You need to create a stock look-up table, the same as was needed for the historical stock data.

C#

```
FinChartData finChartData = null;
```

```
ChartCalendar startDate = new ChartCalendar();
ChartCalendar stopDate = new ChartCalendar();

String[] idStrings = { "INTC", "IBM", "TXN", "AMAT", "CSCO", "AAPL","QQQ" };
String[] tickerStrings = { "INTC", "IBM",  "TXN", "AMAT", "CSCO","AAPL", "QQQ" };

stopDate.Add(ChartObj.DAY_OF_YEAR, -1);
startDate.Add(ChartObj.YEAR, -5);
FinYahooURLCurrentDataSource finStockData = new FinYahooURLCurrentDataSource();
FinYahooURLHistoricalDataSource finStockHistoricalData = new
FinYahooURLHistoricalDataSource();

for (int i = 0; i < idStrings.Length; i++)
{
    finStockHistoricalData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
    finStockData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
}
finChartData = new FinChartData(finStockHistoricalData, finStockData, idStrings,
startDate, stopDate);

InitFinChartView(finChartData);
```

VB

```
Dim finChartData As FinChartData = Nothing
Dim startDate As New ChartCalendar()
Dim stopDate As New ChartCalendar()

Dim idStrings As [String]() = {"INTC","IBM","TXN","AMAT","CSCO","AAPL", "QQQ"}
Dim tickerStrings As [String]() = {"INTC","IBM","TXN","AMAT","CSCO","AAPL","QQQ"}

stopDate.Add(ChartObj.DAY_OF_YEAR, -1)
startDate.Add(ChartObj.YEAR, -5)
Dim finStockData As New FinYahooURLCurrentDataSource()
Dim finStockHistoricalData As New FinYahooURLHistoricalDataSource()

For i As Integer = 0 To idStrings.Length - 1
 finStockHistoricalData.AddTickerLookupItem(idStrings(i), tickerStrings(i))
 finStockData.AddTickerLookupItem(idStrings(i), tickerStrings(i))
Next
finChartData = New FinChartData(finStockHistoricalData, finStockData, idStrings,
startDate, stopDate)

InitFinChartView(finChartData)
```

If you want a current data financial chart, call the FinChartData constructor which has a current data source parameter. The stock lookup table specifies all of the stocks you want displayed as rows in the current data table.

You can add all of the supported column items for a given current data source using the AddAllColumnItems method of the FinChartData class.

C#

```
  finChartData.AddAllColumnItems();
```

VB

```
  finChartData.AddAllColumnItems()
```

Or, you can also specify the column items you want displayed. You use the AddColumnItem method for that.

C#

```
finChartData.AddColumnItem("Name");
finChartData.AddColumnItem("PERatio");
finChartData.AddColumnItem("Volume");
finChartData.AddColumnItem("Bid");
```

```
finChartData.AddColumnItem("BidRealTime");
finChartData.AddColumnItem("ChangeFromYearHigh");
finChartData.AddColumnItem("ChangeFromYearLow");
finChartData.AddColumnItem("DividendShare");
finChartData.AddColumnItem("DividendYield");
finChartData.AddColumnItem("EPSEstimateNextYear");
finChartData.AddColumnItem("EarningsShare");
```

VB

```
finChartData.AddColumnItem("Name")
finChartData.AddColumnItem("PERatio")
finChartData.AddColumnItem("Volume")
finChartData.AddColumnItem("Bid")
finChartData.AddColumnItem("BidRealTime")
finChartData.AddColumnItem("ChangeFromYearHigh")
finChartData.AddColumnItem("ChangeFromYearLow")
finChartData.AddColumnItem("DividendShare")
finChartData.AddColumnItem("DividendYield")
finChartData.AddColumnItem("EPSEstimateNextYear")
finChartData.AddColumnItem("EarningsShare")
```

Important Note: The Yahoo and the Quandl current financial data sources have different column items in them. So you must take into account which current data source you are using when using the AddColumnItem method. Yahoo has the following column items available for display:

Code	Indicator Description
Name	Stock Name
Open	Open
DaysHigh	Days High
DaysLow	Days Low
PreviousClose	PreviousClose
Ask	Ask
Bid	Bid
YearLow	Year Low
YearHigh	Year High
AskRealtime	Ask Realtime
BidRealtime	Bid Realtime
Volume	Volume
LastTradePriceOnly	Last Trade Price Only
LastTradeDate	Last Trade Date
LastTradeTime	Last Trade Time
DaysRange	Days Range
PERatio	PE Ratio
PEGRatio	PEG Ratio
PercentChange	Percent Change
Change	Change
Change_PercentChange	Change_Percent Change
YearRange	Year Range
Currency	Currency

ChangeRealtime	Change Realtime
AfterHoursChangeRealtime	After Hours Change Realtime
DividendShare	Dividend per Share
EarningsShare	Earnings per Share
DividendYield	Dividend Yield
MarketCapitalization	Market Capitalization
EBITDA	EBITDA
BookValue	Book Value
PriceSales	Price/Sales
PriceBook	Price/Book
ShortRatio	Shor tRatio
AverageDailyVolume	Average Daily Volume
EPSEstimateCurrentYear	EPS Estimate Current Year
EPSEstimateNextYear	EPS Estimate Next Year
EPSEstimateNextQuarter	EPS Estimate Next Quarter
PriceEPSEstimateCurrentYear	Price EPS Estimate Current Year
PriceEPSEstimateNextYear	Price EPS Estimate Next Year
OneyrTargetPrice	One yr Target Price
ChangeFromYearLow	Change From Year Low
PercentChangeFromYearLow	Percent Change From Year Low
LastTradeRealtimeWithTime	Last Trade Realtime With Time
ChangePercentRealtime	Change Percent Realtime
ChangeFromYearHigh	Change From Year High
PercentChangeFromYearHigh	Percent Change From Year High
LastTradeWithTime	Last Trade With Time
DaysRangeRealtime	Days Range Realtime
FiftydayMovingAverage	Fifty day Moving Average
TwoHundreddayMovingAverage	Two Hundred day Moving Average
ChangeFromTwoHundreddayMovingAverage	Change From Two Hundred day Moving Average
PercentChangeFromTwoHundreddayMovingAverage	Percent Change From Two Hundred day Moving Average
ChangeFromFiftydayMovingAverage	Change From Fifty day Moving Average
PercentChangeFromFiftydayMovingAverage	Percent Change From Fifty day Moving Average
ExDividendDate	Ex Dividend Date
DividendPayDate	Dividend Pay Date
TickerTrend	TickerTrend
DaysValueChange	Days Value Change
DaysValueChangeRealtime	Days Value Change Realtime
Symbol	Symbol

The Quandl financial current data source includes the following items:

Code	Indicator Description
FLOAT	Number of Shares Outstanding
INSIDER	Insider Holdings
CAPEX	Capital Expenditures
NET_MARG	Net Margin
INV_CAP	Invested Capital
P_S	Price to Sales Ratio
ROC	Return on Capital
STOCK_PX	Stock Price
MKT_DE	Market Debt to Equity Ratio
CORREL	Correlation with the Market
PE_FWD	Forward PE Ratio
REV_GRO	Previous Year Growth in Revenues
EBIT_1T	EBIT for Previous Period
DIV	Dividends
EPS_FWD	Forward Earnings Per Share
CHG_NCWC	Change in Non-Cash Working Capital
CASH_FV	Cash as Percentage of Firm Value
INST_HOLD	Institutional Holdings
EFF_TAX	Effective Tax Rate
CASH_ASSETS	Cash as Percentage of Total Assets
FIXED_TOT	Ratio of Fixed Assets to Total Assets
BETA_VL	Value Line Beta
BV_ASSETS	Book Value of Assets
BV_EQTY	Book Value of Equity
FCFF	Free Cash Flow to Firm
CASH_REV	Cash as Percentage of Revenues
MKT_CAP	Market Capitalization
EFF_TAX_INC	Effective Tax Rate on Income
EV_SALES	EV To Sales Ratio
TOT_DEBT	Total Debt
INTANG_TOT	Ratio of Intangible Assets to Total Assets
PE_G	PE to Growth Ratio
REINV_RATE	Reinvestment Rate
BOOK_DC	Book Debt to Capital Ratio
EPS_GRO_EXP	Expected Growth in Earnings Per Share
EV_EBIT	EV to EBIT Ratio
PE_CURR	Current PE Ratio
MKT_DC	Market Debt to Capital Ratio
NCWC_REV	Non-Cash Working Capital as Percentage of Revenues
REV_12M	Trailing 12-month Revenues
REV_GRO_EXP	Expected Growth in Revenues

Code	Indicator Description
REV_TRAIL	Trailing Revenues
ROE	Return on Equity
EV_EBITDA	EV to EBITDA Ratio
EBITDA	Earnings Before Interest Taxes Depreciation and Amortization
BETA	3-Year Regression Beta
DEPREC	Depreciation
EV_SALESTR	EV to Trailing Sales Ratio
EPS_GRO	Growth in Earnings Per Share
P_BV	Price to Book Value Ratio
NET_INC_TRAIL	Trailing Net Income
PE_TRAIL	Trailing PE Ratio
OP_MARG	Pre-Tax Operating Margin
FIRM_VAL	Firm Value
STDEV	3-year Standard Deviation of Stock Price
TRAD_VOL	Trading Volume
CASH	Cash
DIV_YLD	Dividend Yield
REV_LAST	Revenues
NET_INC	Net Income
EV_BV	EV to Book Value Ratio
REINV	Reinvestment Amount
EBIT	Earnings Before Interest and Taxes
EV_CAP	EV to Invested Capital Ratio
PAYOUT	Payout Ratio
HILO	Hi-Lo Risk
ALLFINANCIALRATIOS	All Financial Ratios
SGA	Sales General and Administration Expenses
EV	Enterprise Value
NCWC	Non-Cash Working Capital

How to add data items to the Quandl current financial data.

C#

```
FinQuandlURLCurrentDataSource finStockData = new FinQuandlURLCurrentDataSource();
FinQuandlURLHistoricalDataSource finStockHistoricalData =
        new FinQuandlURLHistoricalDataSource();


for (int i = 0; i < idStrings.Length; i++)
{
   finStockHistoricalData.AddTickerLookupItem(idStrings[i], historicalTickerLookup[i]);
   finStockData.AddTickerLookupItem(idStrings[i], currentDataStockLookup[i]);
}
finChartData = new FinChartData(finStockHistoricalData, finStockData, idStrings,
startDate, stopDate);

  // items are case invariant

 finChartData.AddColumnItem("Date");
 finChartData.AddColumnItem("Float");
```

```
 finChartData.AddColumnItem("Insider");
 finChartData.AddColumnItem("Capex");

 finChartData.AddColumnItem("Net_marg");
 finChartData.AddColumnItem("Inv_cap");
 finChartData.AddColumnItem("P_s");
 finChartData.AddColumnItem("Roc");
 finChartData.AddColumnItem("Stock_px");
 finChartData.AddColumnItem("Mkt_de");
 finChartData.AddColumnItem("Correl");
 finChartData.AddColumnItem("Pe_fwd");
 finChartData.AddColumnItem("Rev_gro");

 finChartData.AddColumnItem("Ebit");
 finChartData.AddColumnItem("Ebit_1t");
 finChartData.AddColumnItem("Ebitda");
 finChartData.AddColumnItem("Eff_tax");
 finChartData.AddColumnItem("Ev");
 finChartData.AddColumnItem("Ev_cap");
 finChartData.AddColumnItem("Ev_salestr");
 finChartData.AddColumnItem("Ev_ebit");
 finChartData.AddColumnItem("Ev_ebitda");
```

VB

```
Dim finStockData As New FinQuandlURLCurrentDataSource()
Dim finStockHistoricalData As New FinQuandlURLHistoricalDataSource()


For i As Integer = 0 To idStrings.Length - 1
 finStockHistoricalData.AddTickerLookupItem(idStrings(i), historicalTickerLookup(i))
 finStockData.AddTickerLookupItem(idStrings(i), currentDataStockLookup(i))
Next
finChartData = New FinChartData(finStockHistoricalData, finStockData, idStrings,
startDate, stopDate)

' items are case invariant

finChartData.AddColumnItem("Date")
finChartData.AddColumnItem("Float")
finChartData.AddColumnItem("Insider")
finChartData.AddColumnItem("Capex")

finChartData.AddColumnItem("Net_marg")
finChartData.AddColumnItem("Inv_cap")
finChartData.AddColumnItem("P_s")
finChartData.AddColumnItem("Roc")
finChartData.AddColumnItem("Stock_px")
finChartData.AddColumnItem("Mkt_de")
finChartData.AddColumnItem("Correl")
finChartData.AddColumnItem("Pe_fwd")
finChartData.AddColumnItem("Rev_gro")

finChartData.AddColumnItem("Ebit")
finChartData.AddColumnItem("Ebit_1t")
finChartData.AddColumnItem("Ebitda")
finChartData.AddColumnItem("Eff_tax")
finChartData.AddColumnItem("Ev")
finChartData.AddColumnItem("Ev_cap")
finChartData.AddColumnItem("Ev_salestr")
finChartData.AddColumnItem("Ev_ebit")
finChartData.AddColumnItem("Ev_ebitda")
```

How to acquire the current data values used in the charts.

You may want to acquire the current data values used in the creation of the charts table. The values are stored internally in the FinChartData data structures. A typical initialization of the FinChartData class is shown below.

```
FinChartData finChartData = null;

ChartCalendar startDate = new ChartCalendar();
ChartCalendar stopDate = new ChartCalendar();

String[] idStrings = { "Intel", "IBM", "Tex Inst", "App Mat", "CSCO", "Apple", "QQQ" };
String[] tickerStrings = { "INTC", "IBM", "TXN", "AMAT", "CSCO", "AAPL", "QQQ" };

FinYahooURLCurrentDataSource finStockData = null;
FinYahooURLHistoricalDataSource finStockHistoricalData = null;

...

stopDate.Add(ChartObj.DAY_OF_YEAR, -1);
startDate.Add(ChartObj.YEAR, -10);
finStockData = new FinYahooURLCurrentDataSource();
finStockHistoricalData = new FinYahooURLHistoricalDataSource();

for (int i = 0; i < idStrings.Length; i++)
{
    finStockHistoricalData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
    finStockData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
}

finChartData =
   new FinChartData(finStockHistoricalData, finStockData, idStrings, startDate, stopDate);


 finChartData.AddColumnItem("Date");
 finChartData.AddColumnItem("Float");
 finChartData.AddColumnItem("Insider");
 finChartData.AddColumnItem("Capex");

 finChartData.AddColumnItem("Net_marg");
 finChartData.AddColumnItem("Inv_cap");
 finChartData.AddColumnItem("P_s");
 finChartData.AddColumnItem("Roc");
 finChartData.AddColumnItem("Stock_px");
 finChartData.AddColumnItem("Mkt_de");
 finChartData.AddColumnItem("Correl");
 finChartData.AddColumnItem("Pe_fwd");
 finChartData.AddColumnItem("Rev_gro");
```

Assuming that the primary chart has already been displayed, an indication that the current data has already been read from the URL or file, you can retrieve the data using code similar to below.

```
String [,] currentdatastrings = finChartData.ReadStockData();
```

The result is a 2D array of string, where the rows and columns are the same as the rows and columns of the data table. The rows represent stocks, and the columns represent the individual column items you configured the chart for. In the example above, you would retrieve the Pe_fwd string (column item index 11) of the stock with the ID “Tex Inst" (idStrings index 2) using the following code.

```
int stockindex = 2;
int columnid = 11;
String Pe_fwd_string = currentdatastrings[stockindex, columnid];
```

Alternatively, you can retrieve the current data items directly from the data source, using the ReadStockData method. In this case you create a StringArray containing the column items you want, and also specify the ID string of the stock you want the column items for, “Intel” in this case.

```
StringArray columitems = new StringArray();

columitems.Add("Name");
columitems.Add("PERatio");
```

```
columitems.Add("Volume");
columitems.Add("Bid");
columitems.Add("BidRealtime");
columitems.Add("ChangeFromYearHigh");
columitems.Add("ChangeFromYearLow");
columitems.Add("DividendShare");
columitems.Add("DividendYield");
columitems.Add("EPSEstimateNextYear");
columitems.Add("EarningsShare");


StringArray dataitems = finStockData.ReadStockData("Intel", columitems);
```

6. Display Stock Data in the Primary Chart

FinChartView

Com.quinncurtis.chart2dnet.ChartView
FinChartView

The FinChartView class is the main view class of software, which combines a primary chart, with multiple technical indicators in secondary charts. It also includes a zoom window which controls zooming and panning of all of the charts and a financial data table. The user can manage all of the windows using dialog boxes with minimal direct programming. Using it, you can manage a portfolio of securities comparing them against one another, and displaying a variety of technical indicators.

The Primary chart of the FinChartView is reserved for plotting OHLC stock data, and a number of technical indicators overlays. Below is a list of the basic options.

- Plot a single stock as a OHLC, Candlestick, Line, OHLC Bar, Candlestick Volume, Mountain plot, Point and Figure, or Renko chart.
- Plot up to three stocks against one another. Usually this is done using line plots.
- The y-axis of the Primary chart plotting area can be scaled using linear, logarithmic, or normalized scaling. Linear is usually used for plotting a single stock which show price movement in a relatively small range. Logarithmic is used when a stock shows a large change in value over time (Apple for example). Normalized is used when comparing stocks which will usually have a different range of values.

There are two ways you can setup the Primary chart for a combination of these options. The first is to explicitly setup the Primary chart in your program. The second is to do a basic setup, and then permit the user to customize the chart using the built-in Primary chart dialog.

Adding Stocks to the Primary Chart

The first thing you must do is attach the FinChartData you created in the previous chapter, to the FinChartView, using the FinChartView method InitFinChartView. This adds a portfolio of stocks you can choose from, to the current view. Typically you will be working in a subclass of the FinChartView class. So the methods of FinChartView are called without a direct class reference, or by using **this** as the class reference.

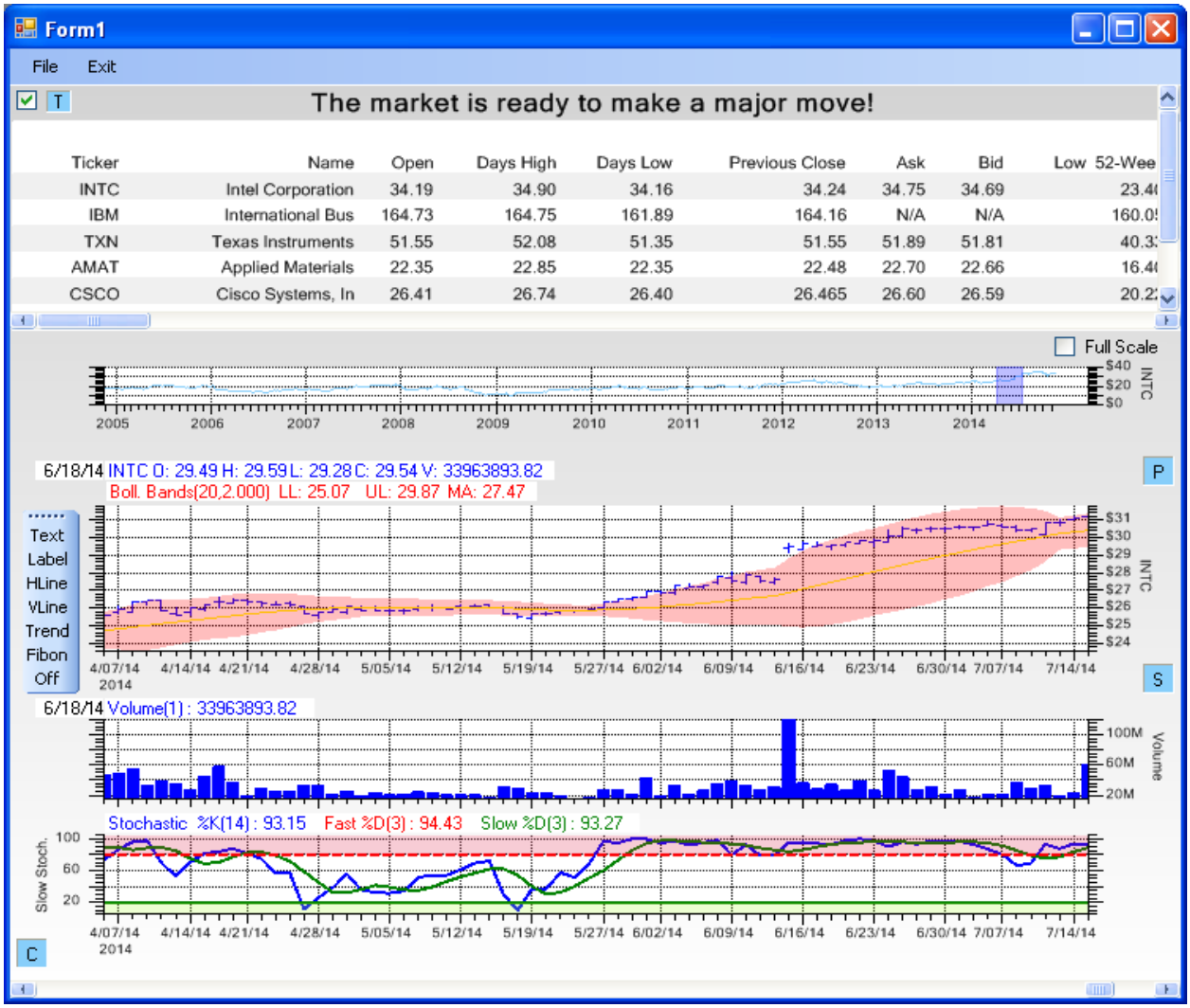
C#

```
this.InitFinChartView(finChartData);
```

VB

```
Me.InitFinChartView(finChartData)
```

Then, add a Primary chart to the view.



The FinChartView can hold one data table, one Zoom chart, one Primary chart, and as many Secondary charts as are practical give the space limitations. With the exception of the Primary chart, all other items are optional.

C#

```
this.FinZoomFlag  = true;
this.AddPrimaryChart("TXN", FinChartConstants.PRIMARYCHART_LINEAR, ChartObj.OHLC);
this.MainTitleString = "Texas Instruments is about to Pop!";
this.AddBollingerBandsToPrimaryChart();
```

VB

```
Me.FinZoomFlag = True
Me.AddPrimaryChart("TXN", FinChartConstants.PRIMARYCHART_LINEAR, ChartObj.OHLC)
Me.MainTitleString ="Texas Instruments is about to Pop!"
Me.AddBollingerBandsToPrimaryChart()
```

The *FinZoomFlag* reference seen above is not needed, because it is set True by default. If you do not want the zoom chart above the Primary chart, set the FinZoomFlag to False. Call AddPrimaryChart, specifying the ticker symbol. The example above specifies "TXN" as the stock to display. This is the id, or key, value in the stock lookup table. So if you entered "Tex. Inst." as the id value in the lookup table, you must use "Tex Inst." in the call to AddPrimaryChart.

The AddPrimaryChart method call adds TXN as the views base plot object. The initial y-axis scale is set to a linear scale. The chart is given the title "Texas Instruments is Ready to Pop!". BollingerBands are an overlay indicator for the primary chart, so the AddBollingerBandsToPrimaryChart call adds Bollinger Bands to the primary chart.

If you want to add a second stock to the chart, use the AddTickerItemToPrimaryChart.

C#
```
this.AddTickerItemToPrimaryChart("INTC");
```

VB
```
Me.AddTickerItemToPrimaryChart("INTC")
```

When you display two or three stocks in a chart using a linear scale and one of the OHLC plot types, it often produces a poor chart. Because the combined range of all three stocks causes the auto-scaling of the y-axis to be so large it obscures the detail in the OHLC values. When comparing different stocks it is better to use a line plot, and a normalized scale.

C#
```
this.AddPrimaryChart("TXN",
    FinChartConstants.PRIMARYCHART_NORMALIZED,  ChartObj.LINE_PLOT);
this.AddTickerItemToPrimaryChart("INTC", ChartObj.LINE_PLOT);
```

VB
```
Me.AddPrimaryChart("TXN",
    FinChartConstants.PRIMARYCHART_NORMALIZED,  ChartObj.LINE_PLOT)
Me.AddTickerItemToPrimaryChart("INTC", ChartObj.LINE_PLOT)
```

In this example, both TXN and INTC have been added to the chart. The plot types have been changed to line plots (ChartObj.LINE_PLOT), and the scale to normalized (FinChartConstants.PRIMARYCHART_NORMALIZED) or logarithmic (FinChartConstants.PRIMARYCHART_LOG) in order to make comparisons easier.

The primary chart, and the technical indicators, by default they are applied to the first of the stocks added to the chart. If you want to explicitly set the current stock to one of the other stocks in the portfolio, set CurrentStockIndex to the index of the stock you want, or

C#
```
this.CurrentStockIndex = 1;
```

VB
```
Me.CurrentStockIndex = 1
```

or set CurrentTickerString to the stock key.

C#
```
this.CurrentTickerString = "INTC";
```

VB
```
Me.CurrentTickerString = "INTC"
```

Simple Moving Average

Simple moving averages (SMA) are an easy way to filter out random noise, or price fluctuations, from a signal. Filtering makes it easier to identify short and long term trends in stock movement. A single simple moving average (MA) is often compared to original signal. When the original signal passes up and through the SMA, that can be considered a buy signal. More details are found in Chapter 2: *Introduction to QCTAChart and Technical Analysis*.

C#

```
public FinSimpleMovingAveragePlot AddSimpleMovingAverageToPrimaryChart(
        int maperiod
)

public FinSimpleMovingAveragePlot AddSimpleMovingAverageToPrimaryChart()
```

VB

```
Public Function AddSimpleMovingAverageToPrimaryChart ( _
        maperiod As Integer _
) As  FinSimpleMovingAveragePlot

Public Function AddSimpleMovingAverageToPrimaryChart As FinSimpleMovingAveragePlot
```

If you use the AddSimpleMovingAverageToPrimaryChart call without parameters, it uses the default values, and the current selected stock.

Parameters

maperiod

Type: Int32
Default Value: 50
period of the moving average

C#

```
double longmaperiod = 50;
double shortmaperiod = 9;
this.AddSimpleMovingAverageToPrimaryChart(longmaperiod);
this.AddSimpleMovingAverageToPrimaryChart(shortmaperiod);
```

VB

```
Dim longmaperiod  As  Double  = 50
Dim  shortmaperiod  As  Double  = 9
Me.AddSimpleMovingAverageToPrimaryChart(longmaperiod)
Me.AddSimpleMovingAverageToPrimaryChart(shortmaperiod)
```

Exponential Moving Average

The exponential moving average (EMA) mathematically weights more recent data more than older data. Because of this it can react quicker to changing circumstances. Otherwise, the buy and sell rules are much the same as the SMA. If the shorter of the EMA signals moves up and through the longer, it considered a buy signal. If the shorter of the EMA signals moves down and through the longer, it considered a sell signal. It can be used singly or in pairs, same as the SMA examples. More details are found in Chapter 2: Introduction to QCTAChart and Technical Analysis.

C#

```
public FinExponentialMovingAveragePlot AddExponentialMovingAverageToPrimaryChart(
        int maperiod
)

public FinExponentialMovingAveragePlot AddExponentialMovingAverageToPrimaryChart()
```

VB

```
Public Function AddExponentialMovingAverageToPrimaryChart ( _
        maperiod As Integer _
) As  FinExponentialMovingAveragePlot

Public Function AddExponentialMovingAverageToPrimaryChart As
FinExponentialMovingAveragePlot
```

Parameters

maperiod
Type: Int32
Default Value: 50
period of the exponential moving average

If you use the AddExponentialMovingAverageToPrimaryChart call without parameters, it uses the default values, and the current selected stock.

C#

```
double longmaperiod = 50;
double shortmaperiod = 9;
this.AddExponentialMovingAverageToPrimaryChart(longmaperiod);
this.AddExponentialMovingAverageToPrimaryChart(shortmaperiod);
```

VB

```
Dim  longmaperiod As Double = 50
Dim shortmaperiod As Double = 9
Me.AddExponentialMovingAverageToPrimaryChart(longmaperiod)
Me.AddExponentialMovingAverageToPrimaryChart(shortmaperiod)
```

Moving Average Bands

Moving Average Bands (or Envelopes) are formed by calculating a SMA (usually a 20-period average) on a source signal, and then forming two bands above and below the SMA signal by adding and subtracting a percentage deviation (usually in the range 1% to 10%) from the SMA signal. Moving average bands serve as an indicator of overbought or oversold conditions, visual representations of price trend, and an indicator of price breakouts. More details are found in Chapter 2: *Introduction to QCTAChart and Technical Analysis*.

C#

```
public FinMABandsPlot AddMABandsToPrimaryChart(
        int maperiod,
        double bandwidth
)

public FinMABandsPlot AddMABandsToPrimaryChart()
```

VB

```
Public Function  AddMABandsToPrimaryChart ( _
        maperiod As Integer, _
        bandwidth As Double _
) As FinMABandsPlot

Public Function AddMABandsToPrimaryChart As FinMABandsPlot
```

If you use the AddMABandsToPrimaryChart call without parameters, it uses the default values, and the current selected stock.

Parameters

maperiod
Type: Int32
Default Value: 20
The moving average period
bandwidth
Type: Double
Default Value: 0.03
The bandwidth value, the percentage value (0.0 to 1.0) above and below to draw the MA bands.

C#
```
double maperiod = 20;
double bandwidth = 0.03;
this.AddMABandsToPrimaryChart(maperiod, bandwidth);
```

VB
```
Dim maperiod As Double = 20
Dim bandwidth As Double = 0.03
Me.AddMABandsToPrimaryChart(maperiod, bandwidth)
```

Bollinger Bands

Bollinger Bands (or Envelopes) are similar to Moving Average bands, except they add a little statistical science to the formation of an indicator. Bollinger Bands still use a SMA calculation for the central line (20 period SMA is standard). But instead of using a fixed percentage as the band width, it defines the separation between the two bands using a multiple of the standard deviation of the signal, calculated using the previous N periods of the closing value of the OHLC data. More details are found in Chapter 2: *Introduction to QCTAChart and Technical Analysis*.

C#

```
public FinBollingerBandsPlot AddBollingerBandsToPrimaryChart(
        int maperiod,
        int smoothing,
        double bandwidth,
        bool fillmode
)
public FinBollingerBandsPlot AddBollingerBandsToPrimaryChart()
```

VB

```
Public Function AddBollingerBandsToPrimaryChart ( _
        maperiod As Integer, _
        smoothing As Integer, _
        bandwidth As Double, _
        fillmode As Boolean _
) As FinBollingerBandsPlot

Public Function AddBollingerBandsToPrimaryChart As FinBollingerBandsPlot
```

If you use the AddBollingerBandsToPrimaryChart call without parameters, it uses the default values, and the current selected stock.

Parameters

maperiod
Type: Int32
Default Value: 20
The period of the moving average to use.
smoothing

Type: Int32
Default Value: FinChartConstants.MA_CALC = 1
The smoothing method to use. Specify FinChartConstants.MA_CALC for a simple moving average, and EXP_MOVING_AVERAGE_TIMEPERIOD

bandwidth
Type: Double
Default Value: 2
The number of standard deviations to use in defining the bands.

fillmode
Type: Boolean
Default Value: true
Set to true to fill the area between the bands.

C#
```
this.AddBollingerBandsToPrimaryChart();
```

VB
```
Me.AddBollingerBandsToPrimaryChart()
```

Paraboloc SAR

The well known market technician J. Welles Wilder created the indicator and described it in his book *New Concepts in Technical Trading Systems*. Published in 1978, the book also describes a number of other Welles indicators, including the Average True Range, the Directional Movement Index and the Relative Strength Index. More details are found in Chapter 2: *Introduction to QCTAChart and Technical Analysis*.

C#

```
public FinParabolicSARPlot AddParabolicSARToPrimaryChart(
        int parsarstartindex,
        double sarstepstart,
        double sarstepincr,
        double sarstepmax
)

public FinParabolicSARPlot AddParabolicSARToPrimaryChart()
```

VB

```
Public Function AddParabolicSARToPrimaryChart ( _
        parsarstartindex As Integer, _
        sarstepstart As Double, _
        sarstepincr As Double, _
        sarstepmax As Double _
) As FinParabolicSARPlot

Public Function AddParabolicSARToPrimaryChart As FinParabolicSARPlot
```

If you use the AddParabolicSARToPrimaryChart call without parameters, it uses the default values, and the current selected stock.

Parameters

parsarstartindex
Type: Int32
Default Value: 5

The Par SAR start index

sarstepstart

Type: Double
Default Value: 0.02
The Par SAR step start value

sarstepincr

Type: Double
Default Value: 0.02
The Par SAR step increment.

sarstepmax

Type: Double
Default Value: 0.2
The Par SAR step max value.

C#

```
int parsarstartindex = 5;
double sarstepstart = 0.02;
double sarstepincr = 0.02;
double sarstepmax = 0.20;
this.AddParabolicSARToPrimaryChart(parsarstartindex, sarstepstart, startstepincr,
sarstepmax);
```

VB

```
Dim parsarstartindex As Integer = 5
Dim sarstepstart As Double = 0.02
Dim sarstepincr As Double = 0.02
Dim sarstepmax As Double = 0.2
Me.AddParabolicSARToPrimaryChart(parsarstartindex, sarstepstart, startstepincr,
sarstepmax)
```

Primary Chart Dialog

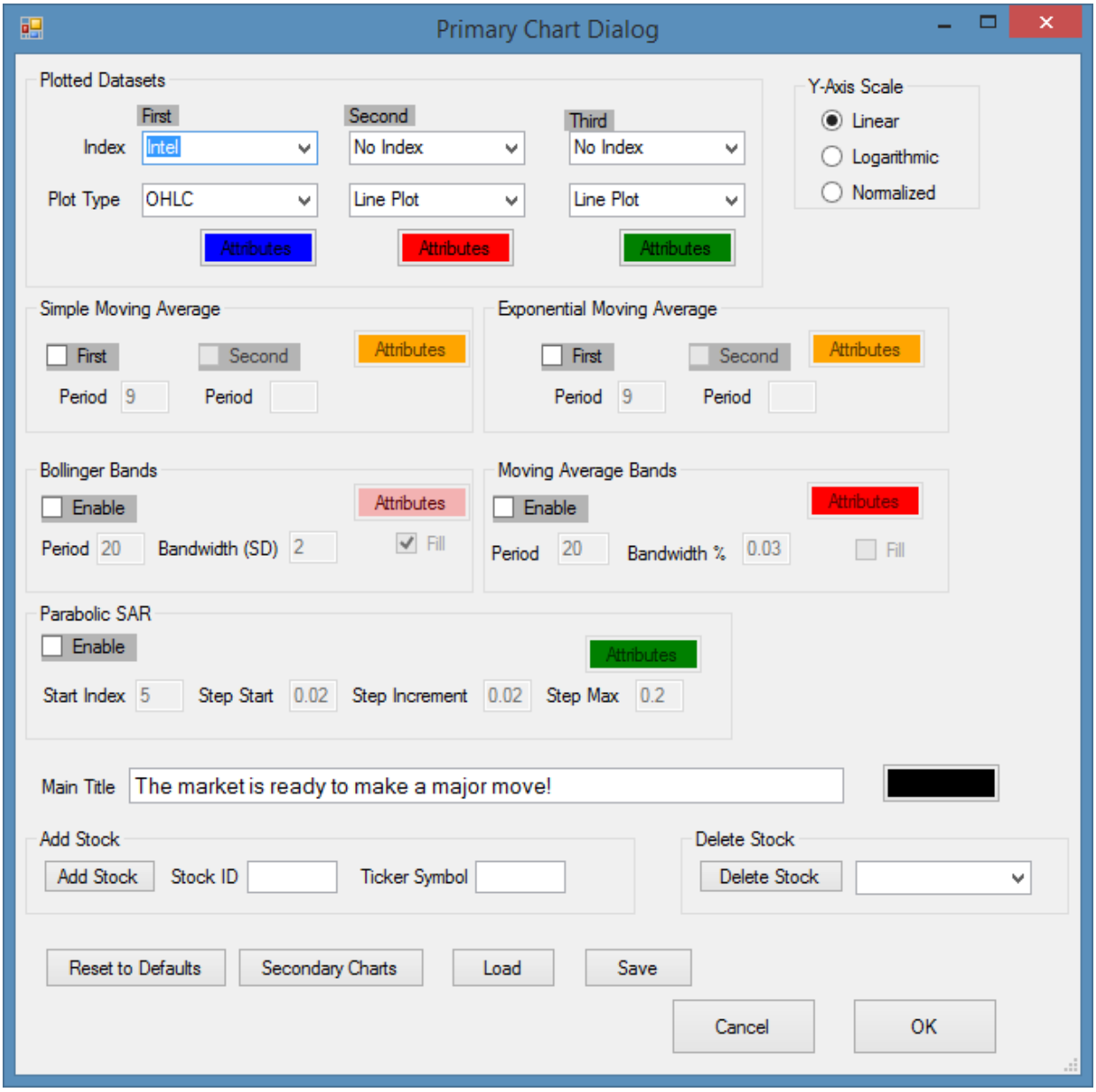
Exactly the same thing can be achieved, without programming, using the Primary Chart Dialog, which is invoked using the P button in the upper right-hand corner of the Primary Chart. Start with a minimum Primary Chart, with just the plot for a single stock, and customize it from there.

C#

```
this.AddPrimaryChart("TXN",  ChartObj.OHLC);
```

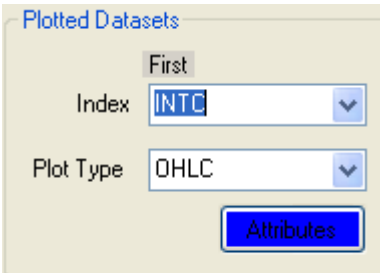
VB

```
Me.AddPrimaryChart("TXN", ChartObj.OHLC)
```



Plotted Datasets

Up to three stocks can be compared in the Primary chart. All technical indicators reference the first of the three stocks.



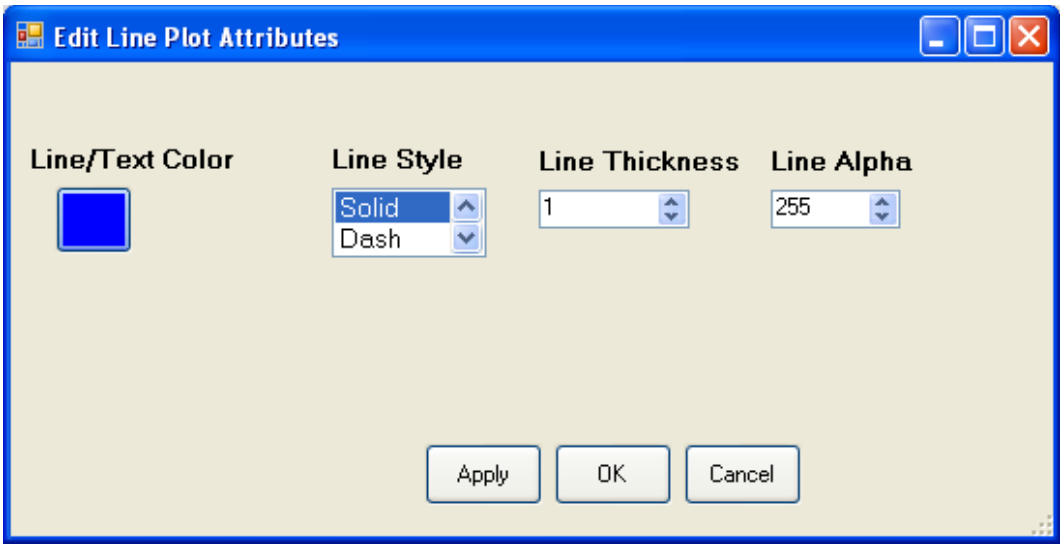
First – First of the plotted stocks

Index – The first plot item in the chart, selected from the available stocks in the attached FinChartData object

Plot Type – The plot type for the first plot – Line, OHLC, OHLC Bar, Candlestick, Candlestick Volume, Mountain, Point and Figure, or Renko

Attributes – Set the attributes of the plot. The actual dialog depends on the **Plot Type**. This includes color, line thickness, fill color, and item width. In the case of the more complex plot types, such as the Point and Figure plot, and the Renko plot, it also includes plot specific items such as pricing mode, box size, reversal count and other parameters.

Line Plot Attributes



Attribute options for line plots

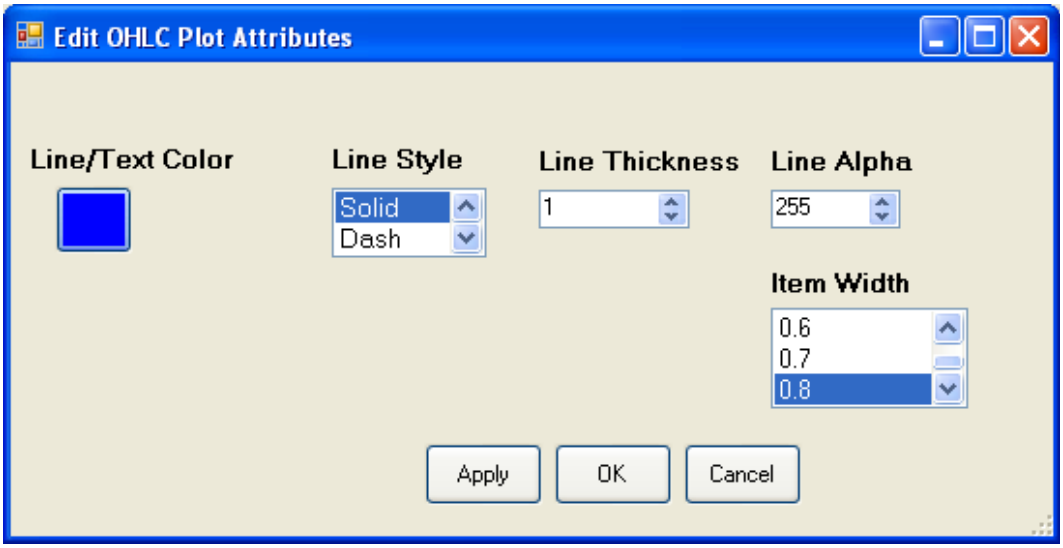
Line/Text Color – Primary color of the line object

Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)

Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Line Alpha – The transparency of the line – use value in the range (0..255)

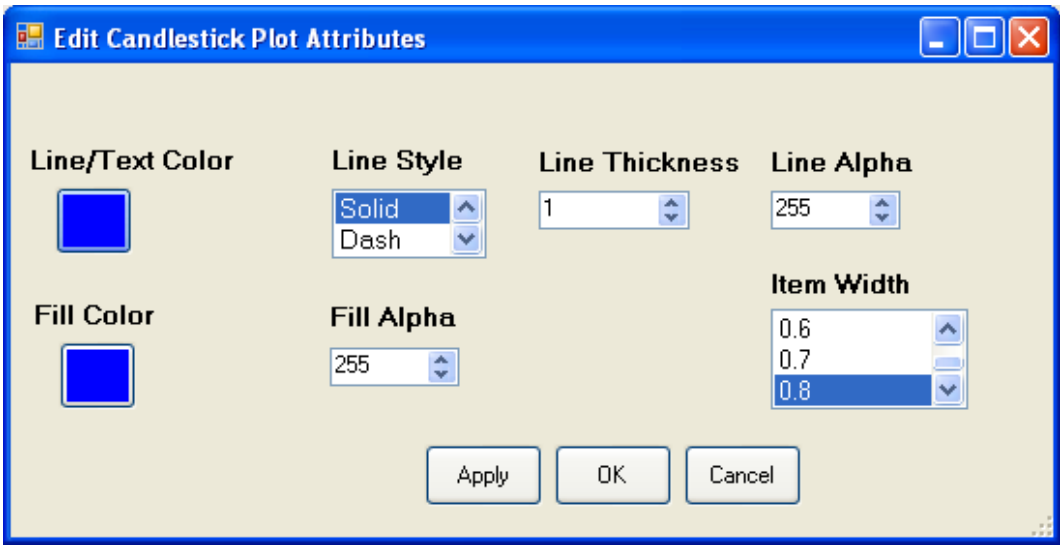
OHLC Attributes



Attribute options for OHLC plots

Line/Text Color – Primary color of the OHLC object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)
Line Alpha – The transparency of the line – use value in the range (0..255)
Item Width – Adjacent values in a plot are separated by a value of 1.0. Enter the item width as a value in the range of (0.0 to 1.0)

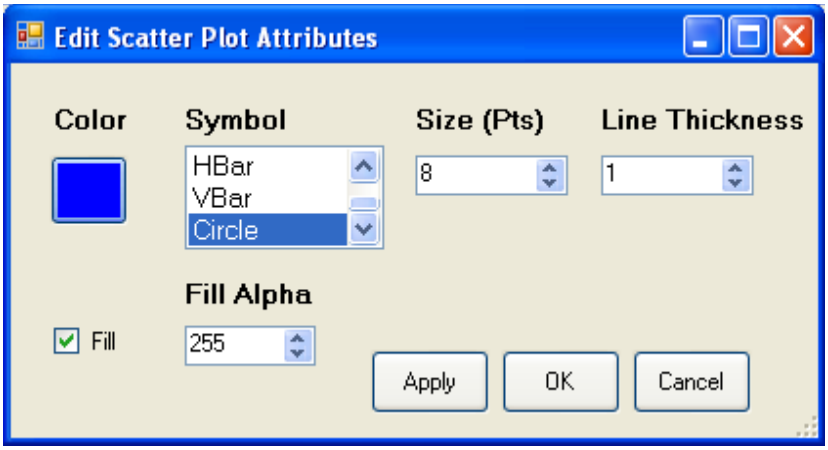
Candlestick, Candelstick Volume, Mountain, Bar Plot Attributes



Attribute options for plots

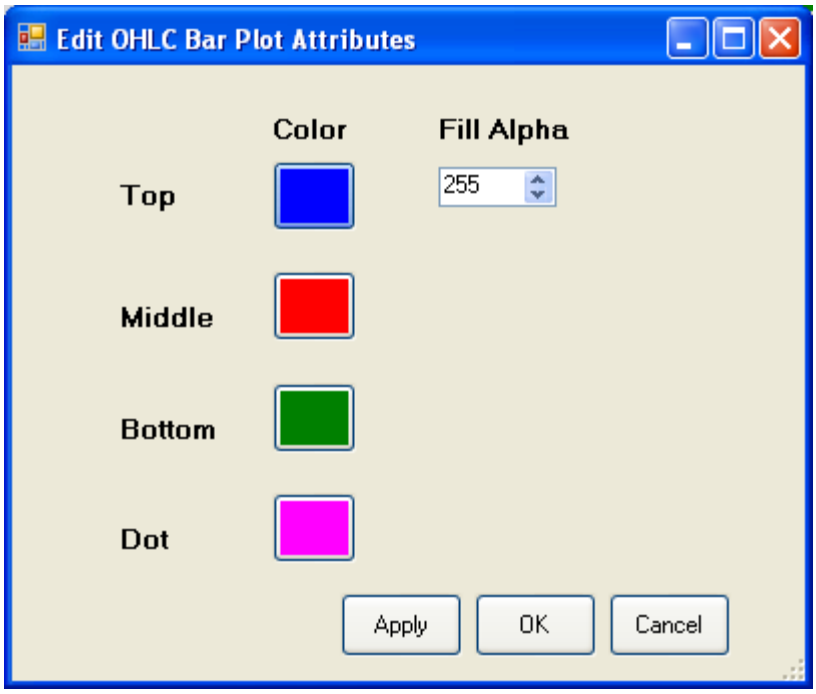
Line/Text Color – Primary line, or outline color of the plot object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)
Line Alpha – The transparency of the line – use value in the range (0..255)
Item Width – Adjacent values in a plot are separated by a value of 1.0. Enter the item width as a value in the range of (0.0 to 1.0)
Fill Color – The color of any fill associated with the object
Fill Alpha – The transparency of the fill color – use value in the range (0..255)

Scatter Plot Attributes



Attribute options for Scatter plots

Color – Primary color of the scatter plot object
Symbol – The symbol used in the scatter plot (Square, Up Triangle, Down Triangle, Diamond, Cross, Plus, Star, Line, Hbar, Vbar, and Circle)
Size (Pts) – Size of the scatter plot symbol in points
Line Width – The width of the line used to draw the scatter plot symbols.
Fill – A flag which if checked specifies that the scatter plot symbol should be filled.
Fill Alpha – The transparency of the fill color – use value in the range (0..255)

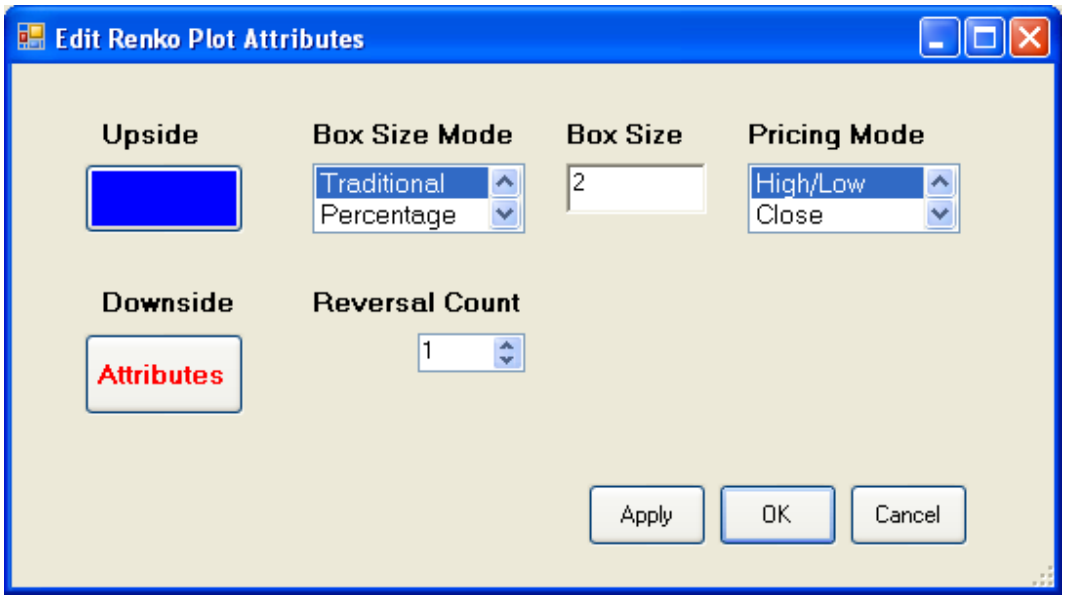


OHLC Bar Attributes

Attribute options for OHLC Bar Attributes

Color – Select the color for the Top, Middle, Bottom, and the Dot.
Fill Alpha – The transparency of the fill color – use value in the range (0..255). The same alpha value is used for the Top, Middle, Bottom and Dot colors.

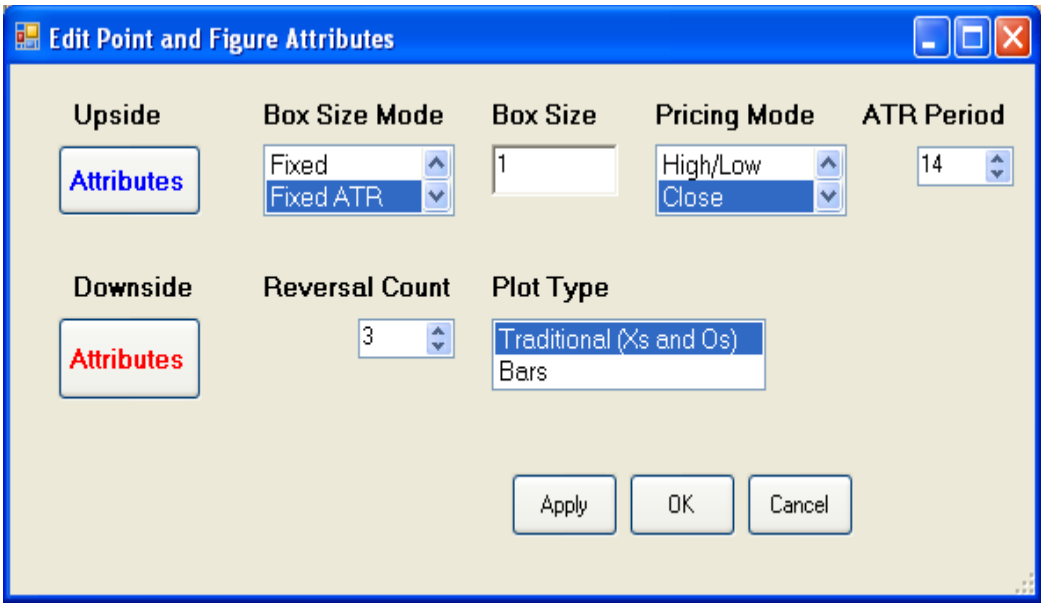
Renko Attributes



Attribute options for Renko plots

Upside - Color used for boxes in an up-trend.
Downside – Color used for boxes in an down-trend.
Box Size – The fixed box size used in calculting and drawing the Renko chart.
Pricing Mode – Selects if the High/Low method or the Close method is used in the Renko chart calculation.
Reversal Count – Specifies the fixed reversal count used in the Renko chart calculations

Point and Figure Attributes

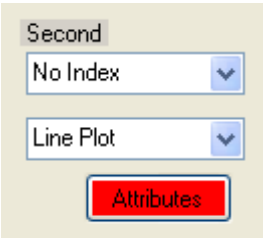


Attribute options for Point and Figure Plots

Upside - Color used for text or boxes in an up-trend.
Downside – Color used for text or boxes in an down-trend.
Box Size Mode – Specifies the box size calculation mode used in calculating the Box size for the Point and

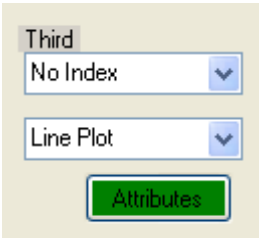
Figure chart – Traditional, Percentage or Fixed, or Fixed ATR.
Box Size – The fixed box size used in calculting and drawing the Point and Figure chart.
Pricing Mode – Selects if the High/Low method or the Close method is used in the Point and Figure chart calculation.
ATR Period – If the **Box Size Mode** is *Fixed ATR*, then the value of **ATR Period** specifies the number of OHLC events used in the calculation.
Reversal Count – Specifies the fixed reversal count used in the Point and Figure chart calculations
Plot Type – Specifies the plot type of the Point and Figure chart – Traditional (Xs and Os), or Bars.

Second – Second of the plotted stocks



Index – The second plot item in the chart, selected from the available stocks in the attached FinChartData object
Plot Type – The plot type for the second plot – Line, OHLC, OHLC Bar, Candlestick, Candlestick Volume or Mountain plot
Attributes – Set the attributes of the plot.

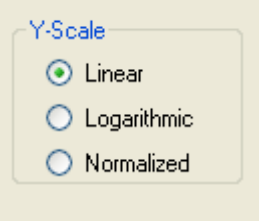
Third – Third of the plotted stocks



Index – The third plot item in the chart, selected from the available stocks in the attached FinChartData object
Plot Type – The plot type for the third plot – Line, OHLC, OHLC Bar, Candlestick, Candlestick Volume or Mountain plot
Attributes – Set the attributes of the plot.

Y-Scale

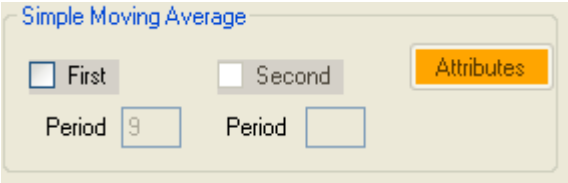
Scaling options for the y-axis



Linear – Set the y-axis scale to linear, best when looking at a single stock with limited range
Logarithmic – Set the y-axis scale to logarithmic, best when looking at a single stock with a wide range
Normalized – Set the y-axis scale to normalized, best when comparing multiple stocks, particularly if they have different ranges.

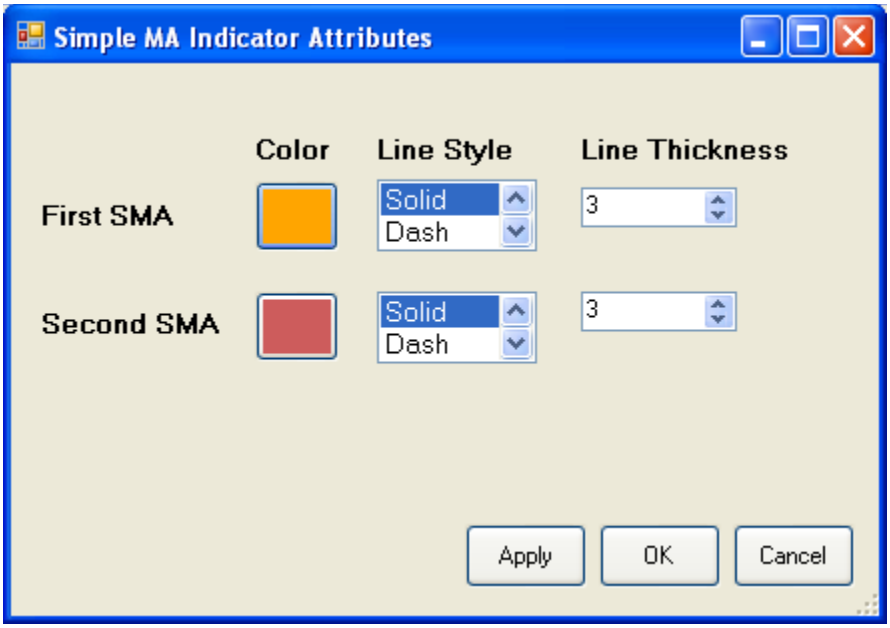
Simple Moving Average

Calculatate and plot as an overlay of the stock plot, one or two simple moving averages of the first plotted dataset.



First – Enable the first simple moving average
Period – The period of the first moving average.
Second– Enable the second simple moving average
Period - The period of the second moving average.
Attributes – Set the attributes of the plot.

Simple Moving Average (SMA) Indicator Attributes



Attribute options for Simple Moving Average (SMA)

First SMA – First of two SMA objects

Second SMA - Second of two SMA objects

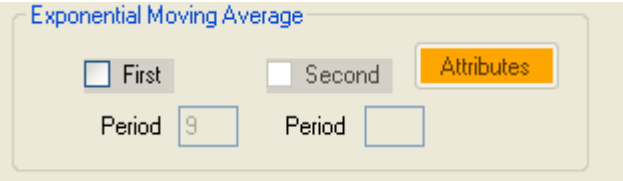
Color – Primary color for the first SMA object

Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)

Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Exponential Moving Average

Calculate and plot as an overlay of the stock plot, one or two exponential moving averages of the first plotted dataset.



First – Enable the first exponential moving average

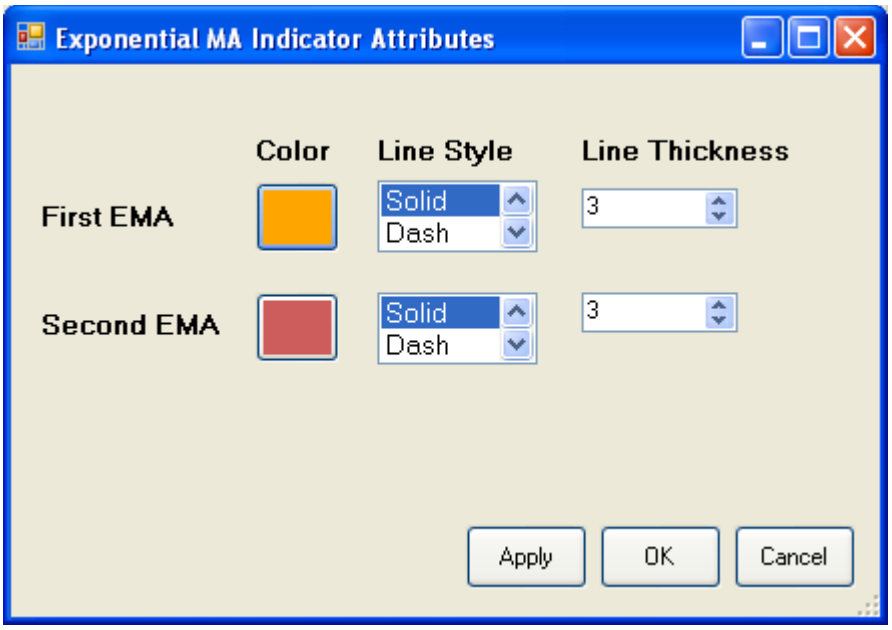
Period – The period of the first exponential average.

Second– Enable the second exponential moving average

Period - The period of the second exponential average.

Attributes – Set the attributes of the plot.

Exponential Moving Average (EMA) Indicator Attributes



Attribute options for Exponential Moving Average (EMA)

First EMA – First of two EMA objects

Second EMA - Second of two SMA objects

Color – Primary color for the first EMA object

Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)

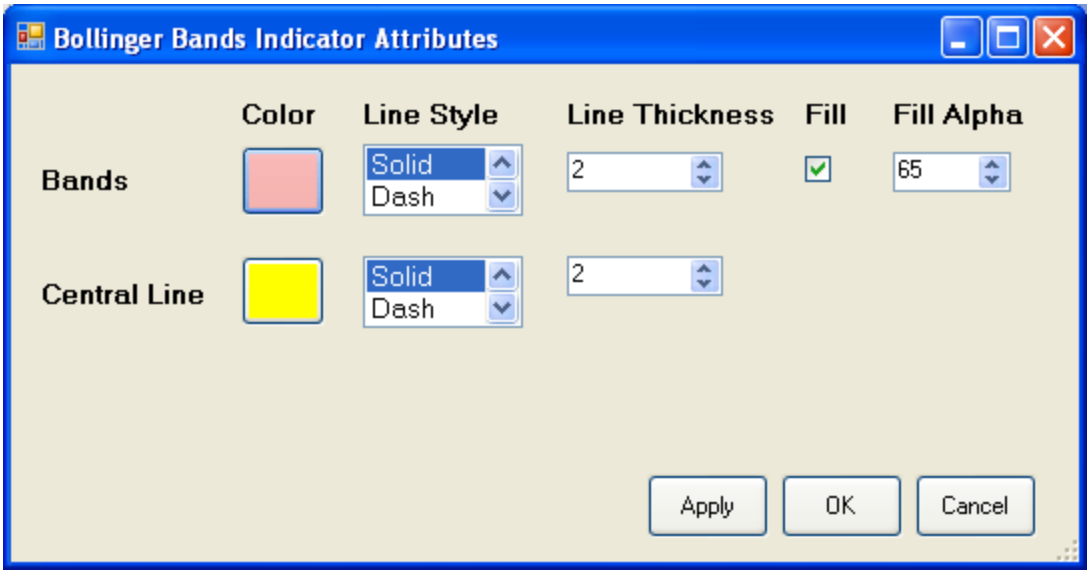
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Bollinger Bands

Options for the Bollinger Bands overlay plot

Enable – Enable the Bollinger Bands plot
Period – The period of the moving average used in the Bollinger Bands smoothing
Bandwidth (SD) – The bandwidth, in Standard Deviations, of the Bollinger Bands plot
Fill – Check and the area between the Bollinger Bands is filled with a trasparent color
Attributes – Set the attributes of the plot.

Bollinger Bands Indicator Attributes



Attribute options for Bollinger Bands

Bands

Color – Primary line, or outline color of the plot object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)
Fill – Check and the area between the Bollinger Bands high and low lines is filled using Color.
Fill Alpha – The transparency of the fill color – use value in the range (0..255)

Central Line

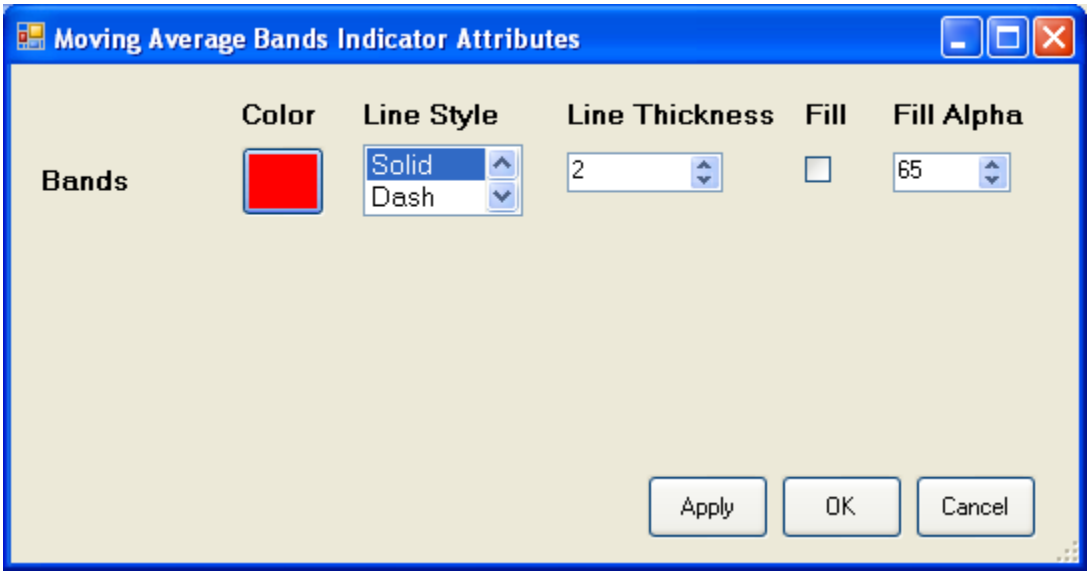
Color – The color of the central line of the Bollinger Bands
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Moving Average Bands

Options for the Moving Average Bands overlay plot

Enable – Enable the Moving Average Bands plot
Period – The period of the moving average used in the Moving Average Bands smoothing
Bandwidth % – The bandwidth, as a percentage of the moving average value of the source signal
Fill – Check and the area between the Moving Average Bands is filled with a trasparent color
Attributes – Set the attributes of the plot.

Moving Average Bands Indicator Attributes



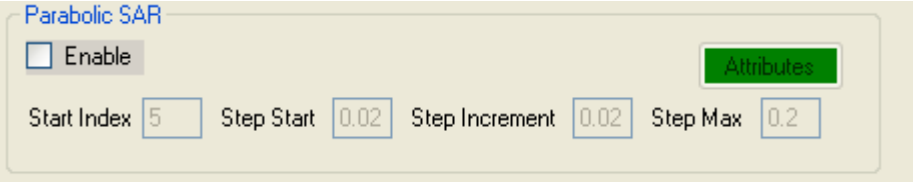
Attribute options for Moving Average Bands

Bands

Color – Primary line, or outline color of the plot object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)
Fill – Check and the area between the moving average high and low lines is filled using Color.
Fill Alpha – The transparency of the fill color – use value in the range (0..255)

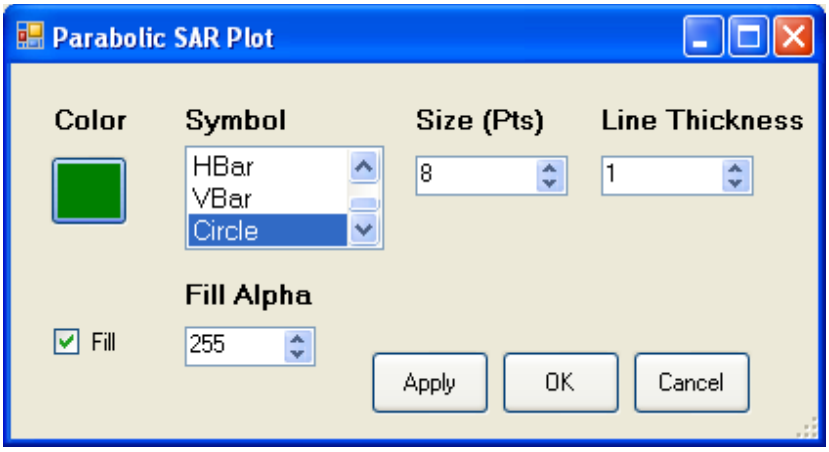
Parabolic SAR

Options for the Parabolic SAR overlay plot



Enable – Enable the Parabolic SAR plot
Start Index – starting index value for the Parabolic SAR plot
Step Start – the starting step size for the for the Parabolic SAR plot
Step Increment – The size of the increment to the step step size when it changes
Step Max – The maximum allowable value of the step size
Attributes – Set the attributes of the plot.

Parabolic SAR Indicator Attributes

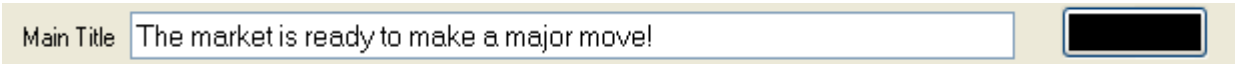


Attribute options for Parabolic SAR

Color – Primary color of the Parabolic SAR plot object
Symbol – The symbol used in the Parabolic SAR (Square, Up Triangle, Down Triangle, Diamond, Cross, Plus, Star, Line, Hbar, Vbar, and Circle(
Size (Pts) – Size of the Parabolic SAR symbol in points
Line Width – The width of the line used to draw the Parabolic SAR symbols.
Fill – A flag which if checked specifies that the Parabolic SAR symbol should be filled.
Fill Alpha – The transparency of the fill color – use value in the range (0..255)

Main Title

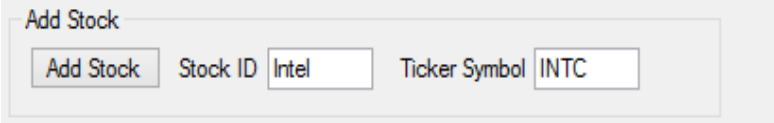
The main title of the chart. It is displayed above the Primary Chart, Zoom window, and the current data table. If the title is empty, the space for the title is used by the charts.



Main Title Text Color – The button to the right of the Main Title text box. Click to change the color of the title text.

6. Display Stock Data in the Primary Chart

Add Stock Option

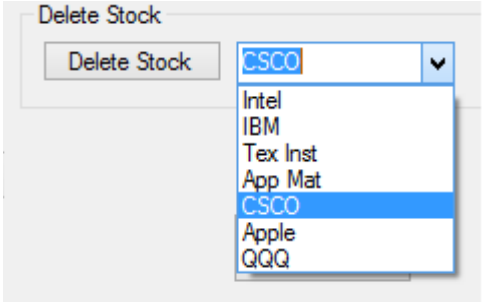


Add Stock – You can add a new stock to the attached stocks by entering in a Stock ID value and a Ticker Symbol.

Stock ID – The string you wish to be used in the display of the stock: "Intel" for example, without the quotes.
Ticker Symbol – The acual ticker symbol used by the exchange: "INTC" for Intel, without the quotes.

Press the **Add Stock** Button once the Stock ID and Ticker Symbol are entered.

Delete Stock Option



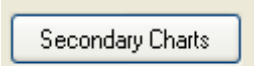
Select a one of current stocks in the drop down combo box list, then select the Delete Stock button. The stock will be removed from all of the stock lists. If the deleted stock is the primary stock (First) in the Plotted Datasets section, the current stock referenced by that list will revert back to the first stock in the First Plotted Datasets combo box. If the deleted stock is reference the Second or Third Plotted Datasets, those lists will revert back to the No Index selction in the associated combo box. Once only one stock is left, you will not be able to delete the final stock; there must always be at least one stock available for selection.

Reset to Defaults

Reset the dialog to its defaults.

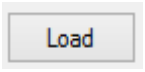
Secondary Charts

Open up the Secondary charts dialog.



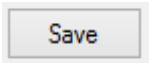
Load

Invoke an Open dialog in order to load a previously saved setup file.



Save

Invoke a Save dialog in order to save the current setup to a file.



Cancel – Close dialog and cancel any changes

OK – Close dialog and apply changes to the chart.

7. Secondary Chart Options

FinChartView

Com.quinncurtis.chart2dnet.ChartView
FinChartView

Underneath the primary chart you can display additional, technical indicator, charts. The secondary charts are used to display technical indicators which are best displayed in their own chart area, usually because they use a y-axis coordinate system which is not the same as the primary chart. The secondary charts always key on the first stock in the primary chart, so you cannot monitor one stock in the Primary chart (INTC for example) and another in the secondary charts (TXN for example).

Many of the secondary chart technical indicators display multiple lines, or bar plots, as part of their display. Also, many use explicit alarm limits to notify the user that a buy or sell signal is taking place.

The secondary indicators are extensively described in Chapter 2. Below is a current list of the secondary indicators available for display in the software.

- Average Directional Indicator
- Momentum
- Rate of Change (ROC)
- Relative Strength (RSI)
- Stochastic (Fast and Slow)
- Williams %R
- Moving Average Convergence/Divergence (MACD)
- Volume charts

There are two ways you can setup secondary charts for a combination of these indicators. The first is to explicitly setup the Secondary charts in your program. The second is to do a basic setup, and then permit the user to customize the chart using the built-in Secondary chart dialog.

Adding Secondary Charts to the FinChartView

The first thing you must do is add the Primary chart to the FinChartView. This is described in the previous chapter. For purposes of this chapter, assume that a simple OHLC plot has been added to the Primary chart, with a Bollinger Bands overlay. The zoom window and the current financial data table is enabled.

C#
```
InitFinChartView(finChartData);
this.EnableFinChartTable = true;
this.FinZoomFlag  = true;
this.AddPrimaryChart("TXN",  ChartObj.OHLC);
```

```
this.MainTitleString = "Texas Instruments is about to Pop!";
this.AddBollingerBandsToPrimaryChart();
```

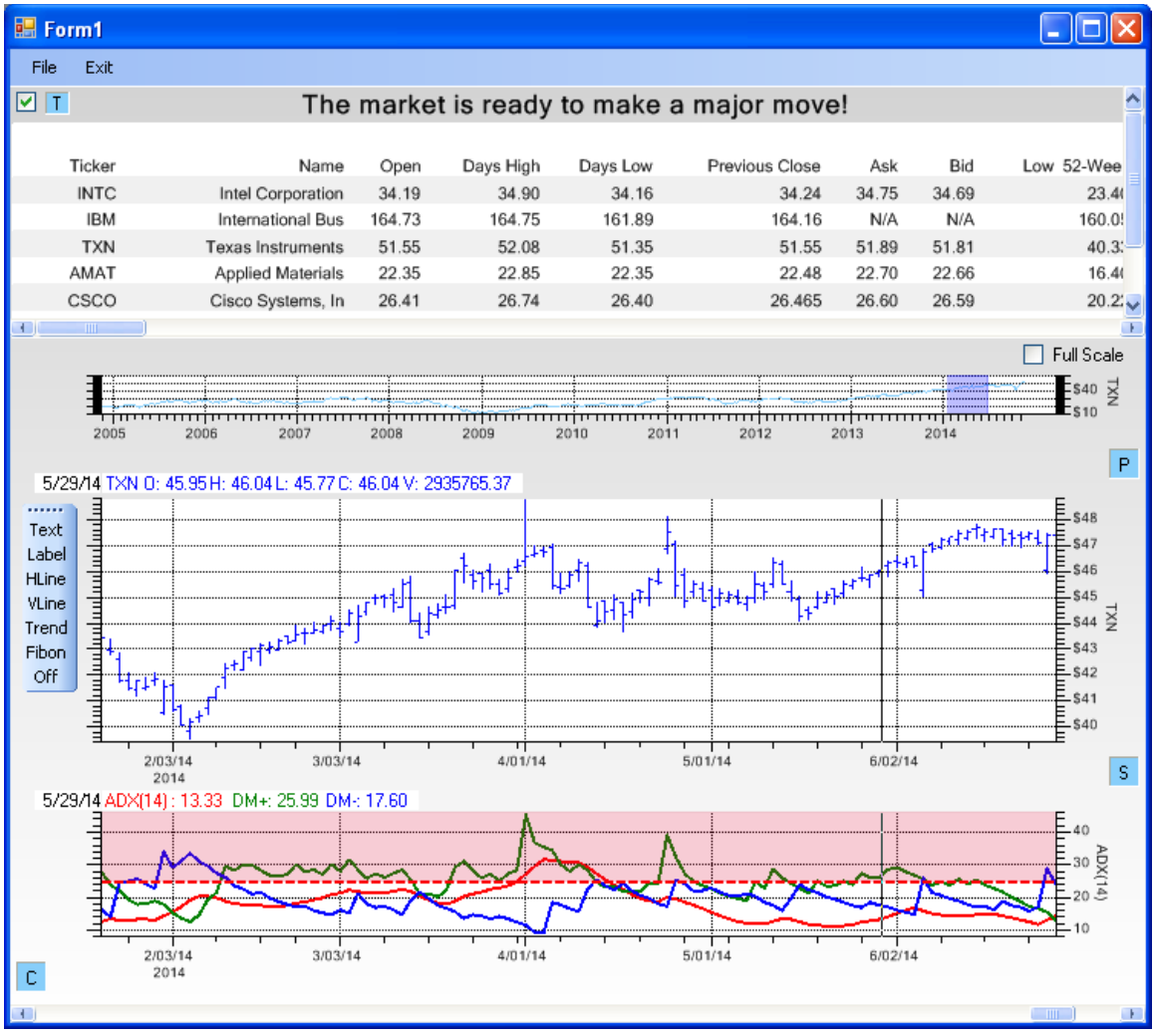
VB

```
InitFinChartView(finChartData)
Me.EnableFinChartTable = True
Me.FinZoomFlag = True
Me.AddPrimaryChart("TXN",  ChartObj.OHLC)
Me.MainTitleString ="Texas Instruments is about to Pop!"
Me.AddBollingerBandsToPrimaryChart()
```

The Primary chart identifies which stock is used for the technical indicators added as secondary charts. Now you can add additional indicators as Secondary charts.

Average Directional Indicator

The Average Directional Indicator (ADX) was originally developed by J. Welles Wilder and is described in his book "New Concepts in Technical Trading System". The ADX is used to measure the strength of a trend. Its related components, the Minus Directional Indicator (-DI) and Plus Directional Indicator (+DI), are used to measure the strength of a trend. The ADX indicator is often used in conjunction with the Parabolic SAR indicator, to confirm a trend. More details are found in Chapter 2: *Introduction to QCTAChart and Technical Analysis*.



C#
```
public FinADXIndicatorPlot AddADXIndicatorChart(
        int stockindex,
        int maperiod,
        int smoothingmode
)

public FinADXIndicatorPlot AddADXIndicatorChart()
```

VB
```
Public Function AddADXIndicatorChart ( _
        stockindex As Integer, _
        maperiod As Integer, _
        smoothingmode As Integer _
) As  FinADXIndicatorPlot

Public Function AddADXIndicatorChart As  FinADXIndicatorPlot
```

Parameters

stockindex
Type: Int32
The stock index.
Default Value: 0

maperiod
Type: Int32
The moving average period

smoothingmode
Type: Int32
The smoothing mode. Use one of the smoothing mode constants: MA_CALC = 1, EXPMA_TIMEPERIOD_CALC = 2, WILDER_EXPMA_CALC

Return Value

Returns the FinADXIndicatorPlot object for the secondary chart.

Limit Value

A stock is said to be in a strong trend if the ADX value is greater than 25 (some use 20 as the threshold). Note, that the ADX indicator does NOT specify a trend direction. So a strong positive trend, or a strong negative trend, will result in ADX values greater than 25 (20). So the ADX value can signal traders whether or not they should be using trend trading strategy. The AddADXIndicatorChart method also places a transparent filled background at the 25 limit level. You can change the default limit level by changing the defaultHighLimitValue static property of the FinADXIndicatorPlot class.

C#
```
FinADXIndicatorPlot.defaultHighLimitValue = 80;

FinADXIndicatorPlot adxchart = this.AddADXIndicatorChart();
```

VB
```
FinADXIndicatorPlot.defaultHighLimitValue = 80
Dim adxchart As FinADXIndicatorPlot = this.AddADXIndicatorChart()
```

Changing the FinADXIndicatorPlot.defaultHighLimitValue property will affect all ADX plots, regardless of

whether they are created programmatically, or using the Secondary chart dialog.

Another way to change the limits, which only affects the specific FinADXIndicatorPlot object is:

C#
```
FinADXIndicatorPlot adxchart = this.AddADXIndicatorChart();
adxchart.HighLimitValue = 80;
```

VB
```
Dim adxchart As FinADXIndicatorPlot = this.AddADXIndicatorChart()
adxchart.HighLimitValue = 80
```

Plot Object Colors

If you want to change the colors of the lines in the ADX chart, use methods found in the FinADXIndicatorPlot class. Change the ADX line atributes using code similar to below.

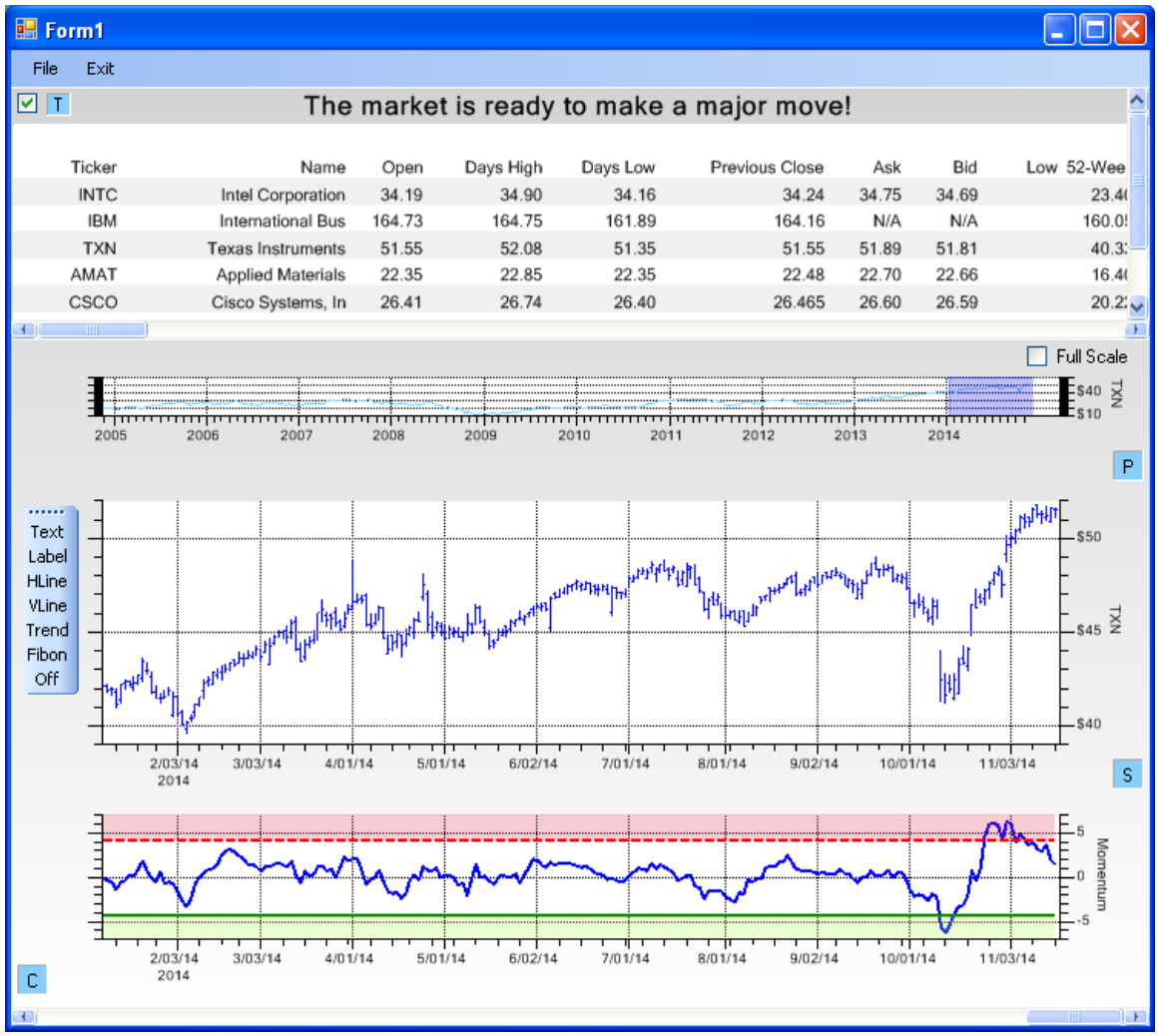
C#

```
FinADXIndicatorPlot adxplot = this.AddADXIndicatorChart();
// ADX line attributes
 adxplot.AdxLinePlotAttribute.PrimaryColor = Color.Red;
 adxplot.AdxLinePlotAttribute.LineWidth = 1;

// ADX DM+ line attributes
 adxplot.AdxDMPLinePlotAttribute.PrimaryColor = Color.Green;
 adxplot.AdxDMPLinePlotAttribute.LineWidth = 1;

// ADX DM- line attributes
 adxplot.AdxDMMLinePlotAttribute.PrimaryColor = Color.Yellow;
 adxplot.AdxDMMLinePlotAttribute.LineWidth = 1;
```

VB
```
Dim adxplot As FinADXIndicatorPlot = Me.AddADXIndicatorChart()
' ADX line attributes
adxplot.AdxLinePlotAttribute.PrimaryColor = Color.Red
adxplot.AdxLinePlotAttribute.LineWidth = 1

' ADX DM+ line attributes
adxplot.AdxDMPLinePlotAttribute.PrimaryColor = Color.Green
adxplot.AdxDMPLinePlotAttribute.LineWidth = 1

' ADX DM- line attributes
adxplot.AdxDMMLinePlotAttribute.PrimaryColor = Color.Yellow
adxplot.AdxDMMLinePlotAttribute.LineWidth = 1
```

Momentum

The Momentum indicator is used to identify the speed (or strength) of a price movement. It is calculated as the difference between today's closing price and the close N days ago. More details are found in Chapter 2: *Introduction to QCTAChart and Technical Analysis*.



C#

```
public FinMomentumIndicatorPlot AddMomentumIndicatorChart(
        int maperiod
)


public FinMomentumIndicatorPlot AddMomentumIndicatorChart()
```

VB

```
Public Function AddMomentumIndicatorChart ( _
        maperiod As Integer _
) As FinMomentumIndicatorPlot

Public Function AddMomentumIndicatorChart As FinMomentumIndicatorPlot
```

Parameters

maperiod
Type: Int32
Default Value: 10
The moving average period

Return Value

Returns the FinMomentumIndicatorPlot object for the secondary chart.

```
FinMomentumIndicatorPlot momentumchart = this.AddMomentumIndicatorChart();
```

Limit Values

Set high (overbought) and low (oversold) thresholds for action. No firm algorithm is available for this. Some suggest setting the thresholds at 67% the high and low peak values on both sides of 0.0. The AddMomentumIndicatorChart method places a transparent filled background at theses levels. You can change the default limit level using the FinMomentumIndicatorPlot.defaultLowLimitFraction and FinMomentumIndicatorPlot.defaultHighLimitFraction static properties, as shown below..

C#
```
FinMomentumIndicatorPlot.defaultLowLimitFraction = 0.55;
FinMomentumIndicatorPlot.defaultHighLimitFraction = 0.55;
FinMomentumIndicatorPlot momentumchart = this.AddMomentumIndicatorChart();
```

VB
```
FinMomentumIndicatorPlot.defaultLowLimitFraction = 0.55
FinMomentumIndicatorPlot.defaultHighLimitFraction = 0.55
Dim momentumchart As FinMomentumIndicatorPlot = Me.AddMomentumIndicatorChart()
```

Changing the static limit properties will affect all Momentum plots, regardless of whether they are created programmatically, or using the Secondary chart dialog.

Another way to change the limits, which only affects the specific FinMomentumIndicatorPlot object is:

C#
```
FinMomentumIndicatorPlot momentumchart = this.AddMomentumIndicatorChart();
momentumchart.HighLimitFraction  = 0.55;
momentumchart.LowLimitFraction  = 0.55;
```

VB
```
Dim momentumchart As FinMomentumIndicatorPlot = Me.AddMomentumIndicatorChart()
momentumchart.HighLimitFraction  = 0.55
momentumchart.LowLimitFraction  = 0.55
```

Plot Object Colors

If you want to change the colors of the lines in the Momentum chart, use methods found in the FinMomentumIndicatorPlot class. Change the Momentum line attributes using code similar to below.

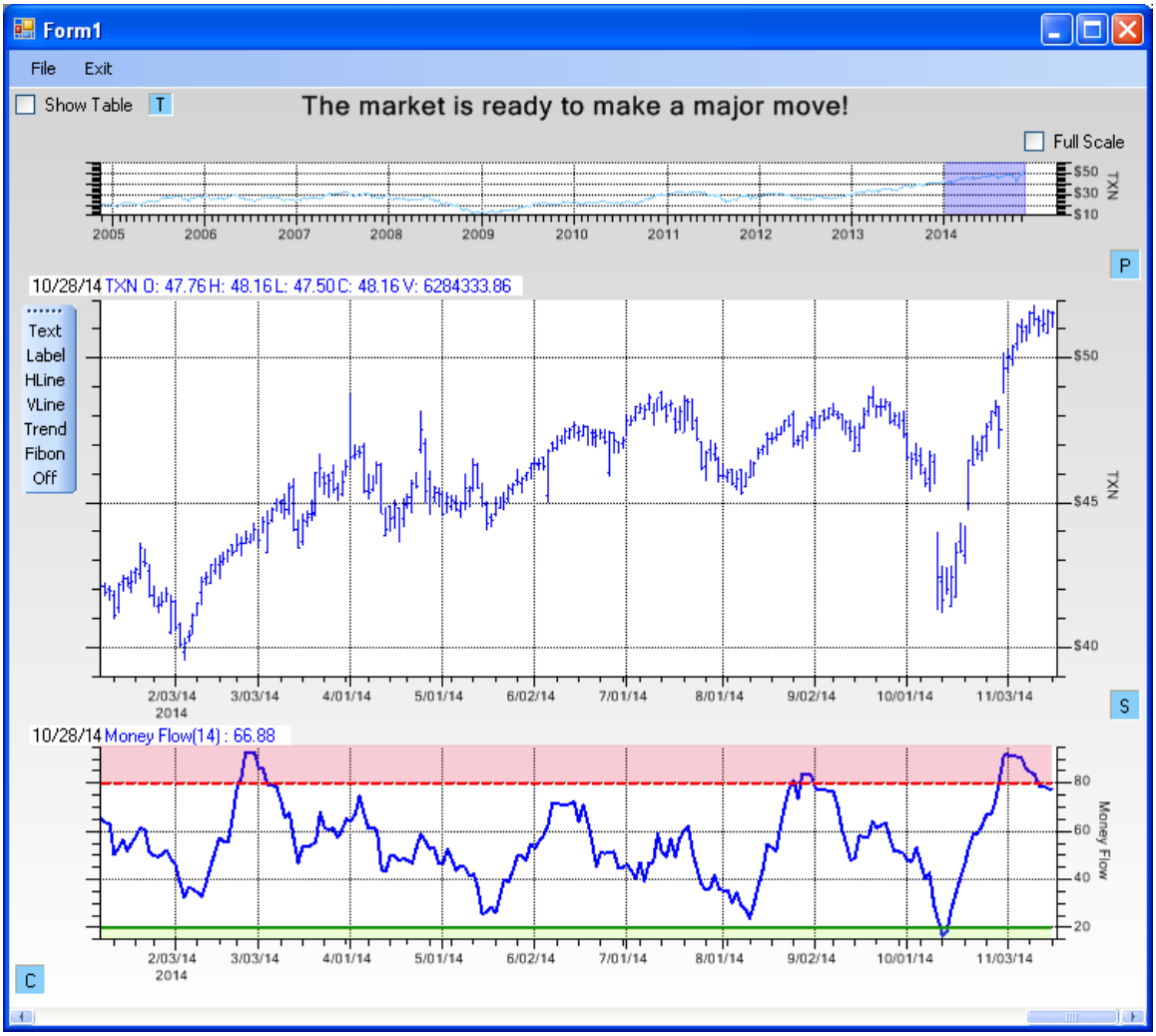
C#
```
FinMomentumIndicatorPlot momentumchart = this.AddMomentumIndicatorChart();
momentumchart.MomentumLinePlotAttribute.PrimaryColor = Color.Purple;
momentumchart.MomentumLinePlotAttribute.LineWidth = 2;
```

VB

```
Dim momentumchart As FinMomentumIndicatorPlot = Me.AddMomentumIndicatorChart()
momentumchart.MomentumLinePlotAttribute.PrimaryColor = Color.Purple
momentumchart.MomentumLinePlotAttribute.LineWidth = 2
```

Money Flow

The Money Flow indicator is considered an oscillator, cycling between 0 and 100. It uses both the Typical Prices for a period, and the period volume, in its calculation. It is sometimes referred to as a volume adjusted RSI indicator. More details are found in Chapter 2: *Introduction to QCTAChart and Technical Analysis*.



C#

```
public FinMoneyFlowIndicatorPlot AddMoneyFlowIndicatorChart(
        int maperiod
)


public FinMoneyFlowIndicatorPlot AddMoneyFlowIndicatorChart()
```

VB

```
Public Function AddMoneyFlowIndicatorChart ( _
```

```
        maperiod As Integer _
) As  FinMoneyFlowIndicatorPlot

Public Function AddMoneyFlowIndicatorChart As  FinMoneyFlowIndicatorPlot
```

Parameters

maperiod

Type: Int32
Default Value: 14
The period used in the Money Flow calculation.

Return Value

Returns the FinMoneyFlowIndicatorPlot object for the secondary chart.

C#

```
FinMoneyFlowIndicatorPlot mfchart = this.AddMoneyFlowIndicatorChart();
```

VB

```
Dim mfchart As FinMoneyFlowIndicatorPlot = Me.AddMoneyFlowIndicatorChart()
```

Limit Values

Set high (overbought) and low (oversold) thresholds for action. No firm algorithm is available for this. Some suggest setting the thresholds at 67% the high and low peak values on both sides of 0.0. The AddMoneyFlowIndicatorChart method places a transparent filled background at theses levels. You can change the default limit level using the FinMoneyFlowIndicatorPlot.defaultLowLimitFraction and FinMoneyFlowIndicatorPlot.defaultHighLimitFraction static properties, as shown below..

C#

```
FinMoneyFlowIndicatorPlot.defaultLowLimitValue = 25;
FinMoneyFlowIndicatorPlot.defaultHighLimitValue = 75;
FinMoneyFlowIndicatorPlot mfchart = this.AddMoneyFlowIndicatorChart();
```

VB

```
FinMoneyFlowIndicatorPlot.defaultLowLimitValue = 25
FinMoneyFlowIndicatorPlot.defaultHighLimitValue = 75
Dim mfchart As FinMoneyFlowIndicatorPlot = Me.AddMoneyFlowIndicatorChart()
```

Changing the static limit properties will affect all Money Flow plots, regardless of whether they are created programmatically, or using the Secondary chart dialog.

Another way to change the limits, which only affects the specific FinMoneyFlowIndicatorPlot object is:

C#

```
FinMoneyFlowIndicatorPlot mfchart = this.AddMoneyFlowIndicatorChart();
mfchart.HighLimitValue = 80;
mfchart.LowLimitValue = 20;
```

VB

```
Dim mfchart As FinMoneyFlowIndicatorPlot = Me.AddMoneyFlowIndicatorChart()
mfchart.HighLimitValue = 80
mfchart.LowLimitValue = 20
```

Plot Object Colors

If you want to change the colors of the lines in the Money Flow chart, use methods found in the FinMoneyFlowIndicatorPlot class. Change the Money Flow line attributes using code similar to below.

C#

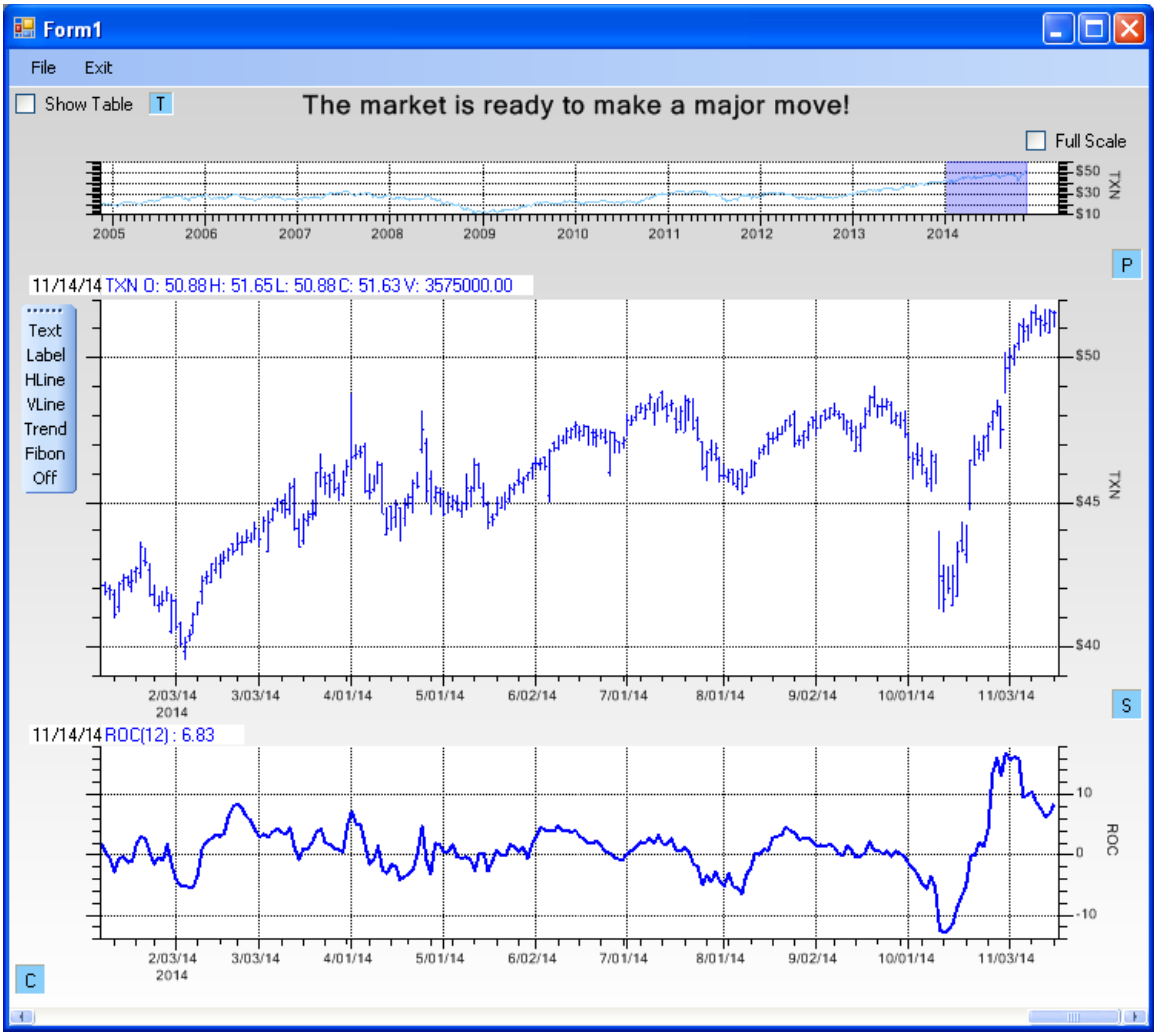
```
FinMoneyFlowIndicatorPlot mfchart = this.AddMoneyFlowIndicatorChart();
mfchart.MoneyFlowLinePlotAttribute.PrimaryColor = Color.Purple;
mfchart.MoneyFlowLinePlotAttribute.LineWidth = 2;
```

VB

```
Dim  mfchart As FinMoneyFlowIndicatorPlot = Me.AddMoneyFlowIndicatorChart()
mfchart.MoneyFlowLinePlotAttribute.PrimaryColor = Color.Purple
mfchart.MoneyFlowLinePlotAttribute.LineWidth = 2
```

Rate of Change (ROC)

The Rate of Change indicator is essentially a normalized version of the Momentum indicator. It takes the Momentum value at each time period and divides it by the Close price N days ago, where N is the same value used in the Momentum calculation. The result is multiplied by 100 to convert from a fraction to a percentage. More details are found in Chapter 2: *Introduction to QCTAChart and Technical Analysis*.



C#
```
public FinRateOfChangeIndicatorPlot AddRateOfChangeIndicatorChart(
        int maperiod
)


public FinRateOfChangeIndicatorPlot AddRateOfChangeIndicatorChart()
```

VB
```
Public Function AddRateOfChangeIndicatorChart ( _
        maperiod As Integer _
) As  FinRateOfChangeIndicatorPlot

Public Function AddRateOfChangeIndicatorChart As  FinRateOfChangeIndicatorPlot
```

Parameters

maperiod
Type: Int32
Default Value: 12
The moving average period

Return Value

Returns the FinRateOfChangeIndicatorPlot object for the secondary chart.

C#

```
FinRateOfChangeIndicatorPlot rocchart = this.AddRateOfChangeIndicatorChart();
```

VB

```
Dim rocchart FinRateOfChangeIndicatorPlot = Me.AddRateOfChangeIndicatorChart()
```

Limit Value

The ROC chart uses 0.0 as the crossover point. When in an up trend, buy when the ROC value passes upward through zero. When in a down trend sell when the ROC value passes downward through zero. No special limit lines are drawn.

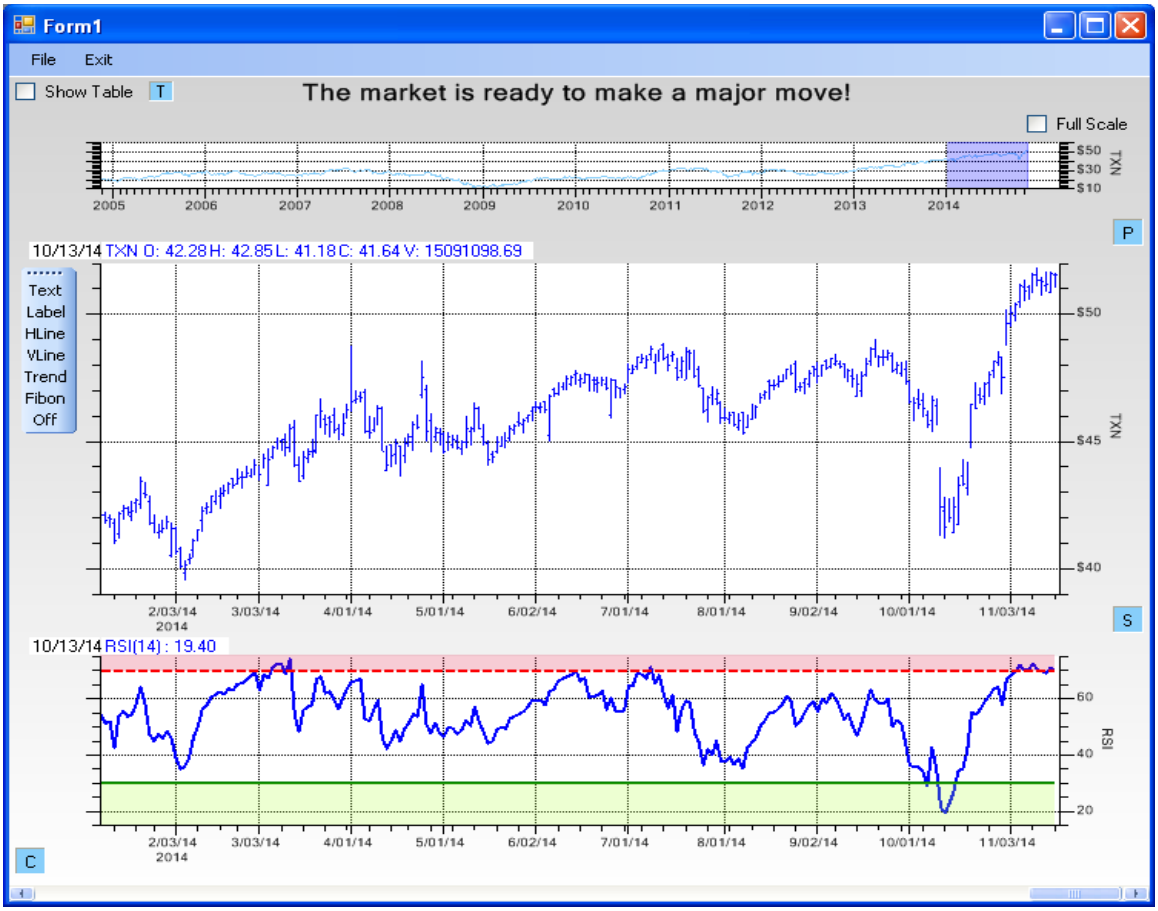
Plot Object Colors

If you want to change the colors of the lines in the ROC chart, use methods found in the FinRateOfChangeIndicatorPlot class. Change the ROC line attributes using code similar to below.

C#
```
FinRateOfChangeIndicatorPlot rocchart = this.AddRateOfChangeIndicatorChart();
rocchart.RateOfChangeLinePlotAttribute.PrimaryColor = Color.Purple;
rocchart.RateOfChangeLinePlotAttribute.LineWidth = 2;
```

VB
```
Dim rocchart As FinRateOfChangeIndicatorPlot = Me.AddRateOfChangeIndicatorChart()
rocchart.RateOfChangeLinePlotAttribute.PrimaryColor = Color.Purple
rocchart.RateOfChangeLinePlotAttribute.LineWidth = 2
```

Relative Strength (RSI)



The Relative Strength Indicator (RSI) was originally developed by J. Welles Wilder and is described in his book "New Concepts in Technical Trading System". It indicates strength or weakness in a security based on the closing price action within the specified trading period. The RSI is calculated using the ratio of higher closes to lower closes: stocks which have had more or stronger positive changes have a higher RSI than stocks which have had more or stronger negative changes. The RSI indicator is normalized so all values are percentages in the range 0 to 100. Intermediate values in the calculation are smoothed using the Wilder exponential smoothing method, usually using a 14 day period. More details are found in Chapter 2: *Introduction to QCTAChart and Technical Analysis*.

C#

```
public FinRSIIndicatorPlot AddRSIIndicatorChart(
        int maperiod
)

public FinRSIIndicatorPlot AddRSIIndicatorChart()
```

VB

```
Public Function AddRSIIndicatorChart ( _
        maperiod As Integer _
) As  FinRSIIndicatorPlot

Public Function AddRSIIndicatorChart As  FinRSIIndicatorPlot
```

Parameters

maperiod
Type: Int32
Default Value: 14
The moving average period

Return Value

Returns the FinRSIIndicatorPlot object for the secondary chart.

Limit Values

Set high (overbought) and low (oversold) thresholds for action limits. Recommended action limits for RSI are a high limit of 70, and a low limit of 30. You can change thedefault limit level using the FinRSIIndicatorPlot.defaultLowLimitValue and FinRSIIndicatorPlot.defaultHighLimitValue static properties, as shown below.

C#
```
FinRSIIndicatorPlot.defaultHighLimitValue = 65;
FinRSIIndicatorPlot.defaultLowLimitValue = 35;
FinRSIIndicatorPlot rsichart = this.AddRSIIndicatorChart();
```

VB
```
FinRSIIndicatorPlot.defaultHighLimitValue = 65
FinRSIIndicatorPlot.defaultLowLimitValue = 35
Dim rsichart As FinRSIIndicatorPlot = Me.AddRSIIndicatorChart()
```

Another way to change the limits, which only affects the specific FinRSIIndicatorPlot object is:

C#
```
FinRSIIndicatorPlot rsichart = this.AddRSIIndicatorChart();
rsichart.HighLimitValue = 65;
rsichart.LowLimitValue = 35;
```

VB
```
Dim rsichart As FinRSIIndicatorPlot = Me.AddRSIIndicatorChart()
rsichart.HighLimitValue = 65
rsichart.LowLimitValue = 35
```

Plot Object Colors

If you want to change the colors of the lines in the RSI chart, use methods found in the FinRSIIndicatorPlot class. Change the RSI line attributes using code similar to below.

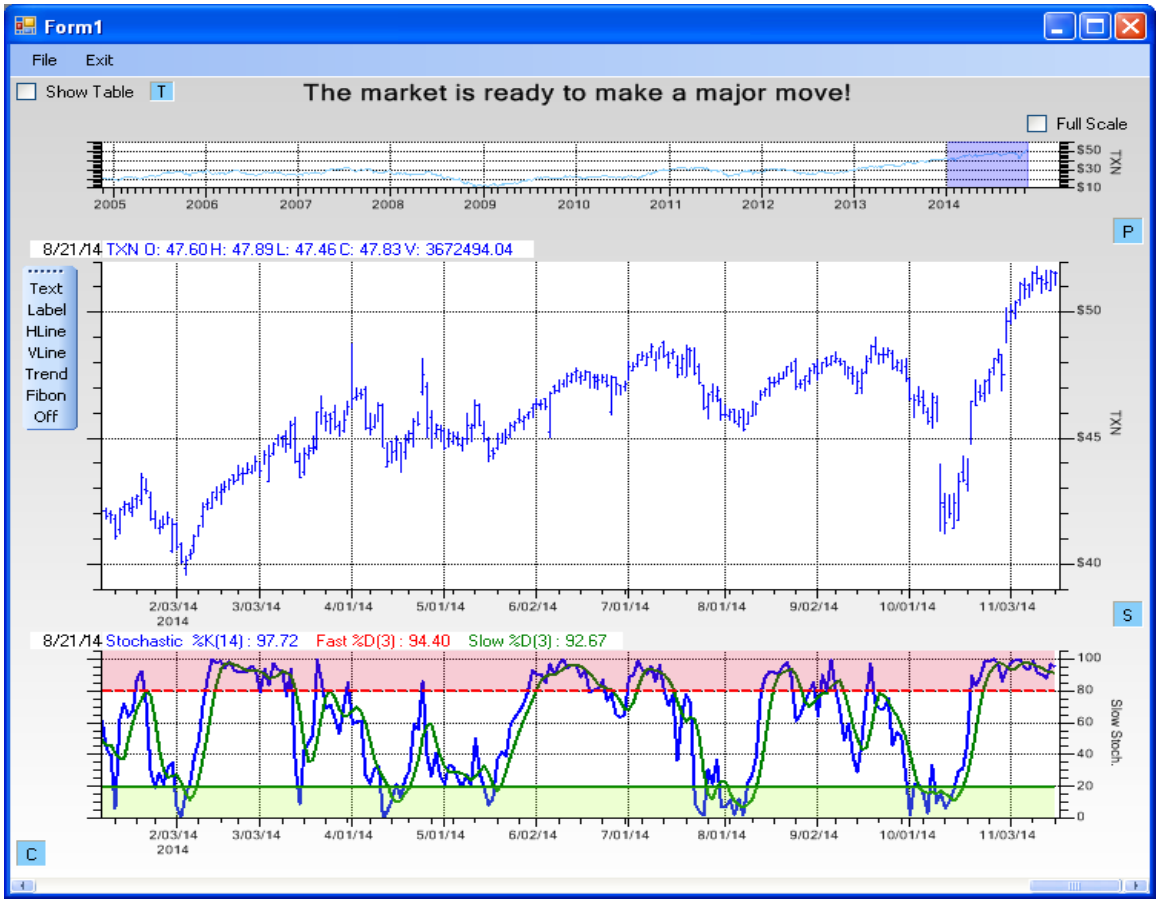
C#
```
FinRSIIndicatorPlot rsichart = this.AddRSIIndicatorChart();
rsichart.RSILinePlotAttribute.PrimaryColor = Color.Purple;
rsichart.RSILinePlotAttribute.LineWidth = 2;
```

VB
```
Dim rsichart As FinRSIIndicatorPlot = Me.AddRSIIndicatorChart()
rsichart.RSILinePlotAttribute.PrimaryColor = Color.Purple
rsichart.RSILinePlotAttribute.LineWidth = 2
```

Stochastic (Fast and Slow)

The Stochastic indicator was developed by George Lane, president of Investment Educators Inc, Watseka, IL. It is based on the assumption that as prices trend upwards, closing prices tend to be in the upper part of the periods

OHLC price range. And the opposite is true, in down trends, closing prices tend to be in the lower part of the periods OHLC price range. More details are found in Chapter 2: *Introduction to QCTAChart and Technical Analysis*.



C#

```
public FinStochasticIndicatorPlot AddStochasticIndicatorChart(
        int mode,
        int maperiod,
        int fastma,
        int slowma
)

public FinStochasticIndicatorPlot AddStochasticIndicatorChart(
        int mode
)

public FinStochasticIndicatorPlot AddStochasticIndicatorChart()
```

VB

```
Public Function AddStochasticIndicatorChart ( _
        mode As Integer, _
        maperiod As Integer, _
        fastma As Integer, _
        slowma As Integer _
) As  FinStochasticIndicatorPlot


Public Function AddStochasticIndicatorChart ( _
```

```
        mode As Integer _
) As  FinStochasticIndicatorPlot

Public Function AddStochasticIndicatorChart As  FinStochasticIndicatorPlot
```

Parameters

mode
Type: Int32
Default: FinChartConstants.STOCHASTIC_OSCILLATOR_SLOW
The type of Stochastic Indicator. Use FinChartConstants.STOCHASTIC_OSCILLATOR_SLOW, or FinChartConstants.STOCHASTIC_OSCILLATOR_FAST

maperiod
Type: Int32
Default: 14
The stochastic K moving average period

fastma
Type: Int32
Default: 3
The fast Stochastic D moving average period

slowma
Type: Int32
Default: 3
The slow Stochastic D moving average period

Return Value

Returns the FinStochasticIndicatorPlot object for the secondary chart.

Limit Values

Set high (overbought) and low (oversold) thresholds for action limits. Recommended action limits for Stochatic charts are a high limit of 80, and a low limit of 20. You can change thedefault limit level using the FinStochasticIndicatorPlot.defaultLowLimitValue and FinStochasticIndicatorPlot.defaultHighLimitValue static properties, as shown below.

C#
```
FinStochasticIndicatorPlot.defaultHighLimitValue = 80;
FinStochasticIndicatorPlot.defaultLowLimitValue = 20;
FinStochasticIndicatorPlot stochchart = this.AddStochasticIndicatorChart();
```

VB
```
FinStochasticIndicatorPlot.defaultHighLimitValue = 80
FinStochasticIndicatorPlot.defaultLowLimitValue = 20
Dim stochchart FinStochasticIndicatorPlot = Me.AddStochasticIndicatorChart()
```

Another way to change the limits, which only affects the specific FinStochasticIndicatorPlot object is:

C#
```
FinStochasticIndicatorPlot stochchart = this.AddStochasticIndicatorChart();
stochchart.HighLimitValue = 65;
stochchart.LowLimitValue = 35;
```

VB
```
Dim stochchart As FinStochasticIndicatorPlot = Me.AddStochasticIndicatorChart()
```

```
stochchart.HighLimitValue = 65
stochchart.LowLimitValue = 35
```

Plot Object Colors

If you want to change the colors of the lines in the Stochastic chart, use methods found in the FinStochasticIndicatorPlot class. Change the Stochastic line atributes using code similar to below.

C#

```
FinStochasticIndicatorPlot stochchart = this.AddStochasticIndicatorChart();

// Stochastic line attributes
 stochchart.StochasticAttribute.PrimaryColor = Color.Red;
 stochchart.StochasticAttribute.LineWidth = 1;

// Fast Stochastic line attributes
 stochchart.StochasticFastAttribute.PrimaryColor = Color.Green;
 stochchart.StochasticFastAttribute.LineWidth = 1;

// Slow Stochastic line attributes
 stochchart.StochasticSlowAttribute.PrimaryColor = Color.Green;
 stochchart.StochasticSlowAttribute.LineWidth = 1;
```

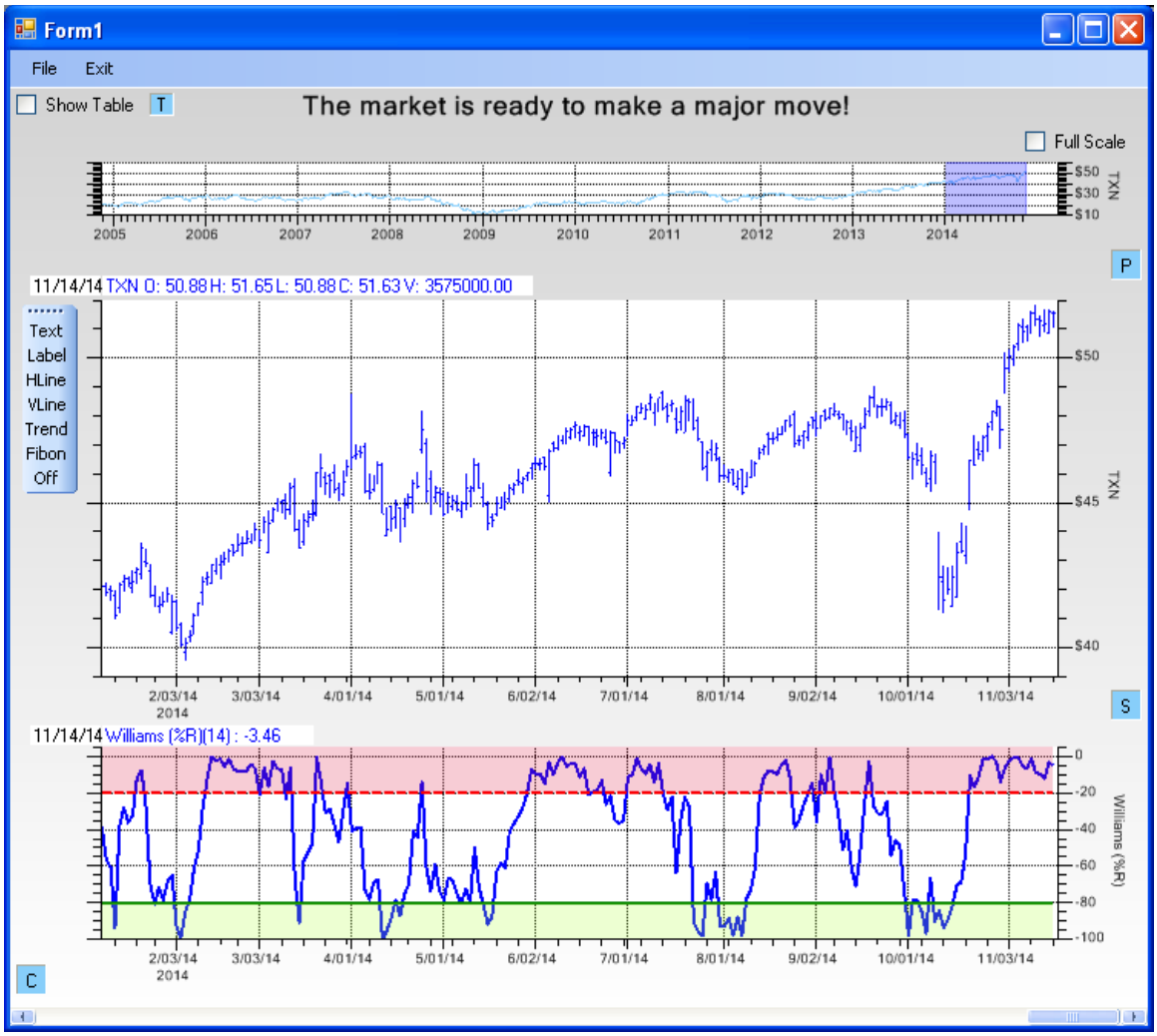
VB

```
Dim stochchart As FinStochasticIndicatorPlot = Me.AddStochasticIndicatorChart()

' Stochastic line attributes
 stochchart.StochasticAttribute.PrimaryColor = Color.Red
 stochchart.StochasticAttribute.LineWidth = 1

'Fast Stochastic line attributes
 stochchart.StochasticFastAttribute.PrimaryColor = Color.Green
 stochchart.StochasticFastAttribute.LineWidth = 1

' Slow Stochastic line attributes
 stochchart.StochasticSlowAttribute.PrimaryColor = Color.Green
 stochchart.StochasticSlowAttribute.LineWidth = 1
```

Williams %R

Williams %R was developed by Larry Williams, a publisher of trading and technical analysis materials, to indicate overbought and oversold market conditions for a stock. Similar to the Stochastic indicator, it compares a stock's close to the high-low range over a certain period of time, usually 14 days. For some strange reason the indicator is normalized in the range 0.0 to -100, so that an overbought condition is represented by the indicator above the -20 line, and an oversold condition represented by the indicator below -80. A value of -100 for the index means that the close today was at the lowest low of the past N days, and a value of 0 for the index means that a close today was at the highest high of the past N days. When it is plotted the Williams %R indicator looks the same as the Stochastic %K indicator, except that the y-axis values are shifted down by 100. More details are found in Chapter 2: *Introduction to QCTAChart and Technical Analysis*.

7. Secondary Chart Options



C#

```
public FinWilliamsRIndicatorPlot AddWilliamsRIndicatorChart(
        int maperiod
)

public FinWilliamsRIndicatorPlot AddWilliamsRIndicatorChart()
```

VB

```
Public Function AddWilliamsRIndicatorChart ( _
        maperiod As Integer _
) As FinWilliamsRIndicatorPlot

Public Function AddWilliamsRIndicatorChart As FinWilliamsRIndicatorPlot
```

Parameters

maperiod

Type: Int32
Default: 14
The moving average period

Return Value

Returns the FinWilliamsRIndicatorPlot object for the secondary chart.

Limit Values

Set high (overbought) and low (oversold) thresholds for action limits. Recommended action limits for Williams %R are a high limit of -20, and a low limit of -80. You can change thedefault limit level using the FinWilliamsRIndicatorPlot.defaultLowLimitValue and FinWilliamsRIndicatorPlot.defaultHighLimitValue static properties, as shown below.

C#
```
FinWilliamsRIndicatorPlot.defaultHighLimitValue = -25;
FinWilliamsRIndicatorPlot.defaultLowLimitValue = -75;
FinWilliamsRIndicatorPlot willimanshart = this.AddWilliamsRIndicatorChart();
```

VB
```
FinWilliamsRIndicatorPlot.defaultHighLimitValue = -25
FinWilliamsRIndicatorPlot.defaultLowLimitValue = -75
Dim willimanshart As FinWilliamsRIndicatorPlot = Me.AddWilliamsRIndicatorChart()
```

Changing the static limit properties will affect all Williams %R plots, regardless of whether they are created programmatically, or using the Secondary chart dialog.

Another way to change the limits, which only affects the specific FinWilliamsRIndicatorPlot object is:

C#
```
FinWilliamsRIndicatorPlot willimanshart = this.AddWilliamsRIndicatorChart();
willimanschart.HighLimitValue = -25;
willimanschart.LowLimitValue = -75;
```

VB
```
Dim willimanshart As FinWilliamsRIndicatorPlot = Me.AddWilliamsRIndicatorChart()
willimanschart.HighLimitValue = -25
willimanschart.LowLimitValue = -75
```

Plot Object Colors

If you want to change the colors of the lines in the Williams %R chart, use methods found in the FinWilliamsRIndicatorPlot class. Change the Williams %R line attributes using code similar to below.

C#
```
FinWilliamsRIndicatorPlot willimanschart = this.AddWilliamsRIndicatorChart();
willimanschart.WilliamsRLinePlotAttribute.PrimaryColor = Color.Purple;
willimanschart.WilliamsRLinePlotAttribute.LineWidth = 2;
```

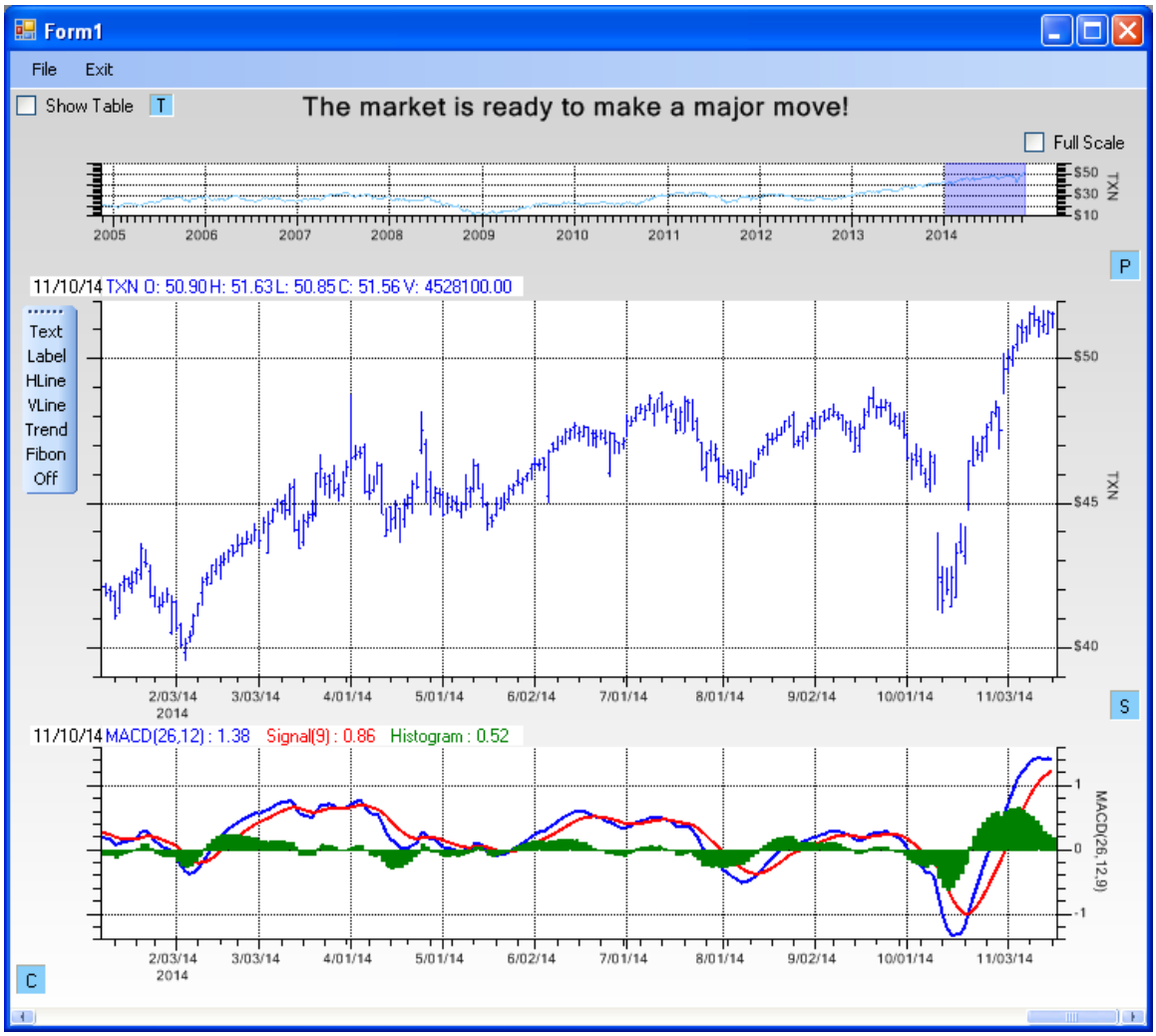
VB
```
Dim willimanschart As FinWilliamsRIndicatorPlot = Me.AddWilliamsRIndicatorChart()
willimanschart.WilliamsRLinePlotAttribute.PrimaryColor = Color.Purple
willimanschart.WilliamsRLinePlotAttribute.LineWidth = 2
```

Moving Average Convergence/Divergence (MACD)

The MACD was developed by Gerald Appel and is discussed in his book, *The Moving Average Convergence Divergence Trading Method*. It is a comparison of several moving averages (exponential) derived from a stocks closing prices.

The first EMA uses a 26 day period and is referred to as the long EMA line. The second EMA uses a 12 day period and is referred to as the short EMA line. The MACD line is calculated by subtracting the long (26) EMA line from

the short (12) EMA line. The signal line is calculated as a 9 day EMA of the MACD line. The two lines actually plotted in the indicator are the MACD line, and the signal line. The crossover of MACD and signal lines indicates a buying or selling opportunity. Additionally, a histogram, representing the differences between the MACD line, and the signal line, is usually part of a MACD chart. More details are found in Chapter 2: *Introduction to QCTAChart and Technical Analysis*.



C#

```
public FinMACDIndicatorPlot AddMACDIndicatorChart(
        int macdlong,
        int macdshort,
        int signalperiod,
        int smoothingmode
)

public FinMACDIndicatorPlot AddMACDIndicatorChart()
```

VB

```
Public Function AddMACDIndicatorChart ( _
        macdlong As Integer, _
        macdshort As Integer, _
        signalperiod As Integer, _
        smoothingmode As Integer _
```

```
) As FinMACDIndicatorPlot

Public Function AddMACDIndicatorChart As FinMACDIndicatorPlot
```

Parameters

macdlong
Type: Int32
Default: 26
The long moving average period
macdshort
Type: Int32
Default: 12
The short moving average period.
signalperiod
Type: Int32
Default: 9
The signal period.
smoothingmode
Type: Int32
Default: FinChartConstants.EXP_MOVING_AVERAGE_TIMEPERIOD
Use one of the smoothing mode constants: FinChartConstants.MA_CALC for a simple moving average, and EXP_MOVING_AVERAGE_TIMEPERIOD for an exponential moving average.

Return Value

Returns the FinMACDIndicatorPlot object for the secondary chart.

Limit Values

There are no limit lines for a MACD indicator chart.

Plot Object Colors

If you want to change the colors of the lines in the MACD chart, use methods found in the MACD class. Change the MACD line attributes using code similar to below.

C#

```
// MACD line attribute
FinMACDIndicatorPlot macdchart = this.FinMACDIndicatorPlot();
macdchart.MACDAttribute.PrimaryColor = Color.Purple;
macdchart.MACDAttribute.LineWidth = 2;

// MACD signal line attribute
macdchart.MACDSignalAttribute.PrimaryColor = Color.Purple;
macdchart.MACDSignalAttribute.LineWidth = 2;

// MACD histogram attribute
macdchart.MACDHistogramAttribute.PrimaryColor = Color.Purple;
macdchart.MACDHistogramAttribute.LineWidth = 2;
```

VB

```
' MACD line attribute
Dim macdchart As FinMACDIndicatorPlot = Me.FinMACDIndicatorPlot()
macdchart.MACDAttribute.PrimaryColor = Color.Purple
macdchart.MACDAttribute.LineWidth = 2

' MACD signal line attribute
macdchart.MACDSignalAttribute.PrimaryColor = Color.Purple
```

```
macdchart.MACDSignalAttribute.LineWidth = 2

' MACD histogram attribute
macdchart.MACDHistogramAttribute.PrimaryColor = Color.Purple
macdchart.MACDHistogramAttribute.LineWidth = 2
```

Volume charts

Volume represents the number of shares, or contracts, which traded during a period of time. Since it is part of the standard OHLCV price packet available in historical data feeds, it gives valuable information about the significance of the price action of a stock. A price movement in either direction is considered more relevant if is is simultaneous with a large increase in share volume. If you are monitoring the trend of a stock, and that stock suddenly reverses the trend, that is the time to check the volume to see if the reversal was accompanied by a sharp increase in the trading volume. If not, then that can be a sign that there is no conviction by the trend reversal, and you can expect the stock to reverse back. Volume charts can be smoothed using the standard smoothing techniques to help filter out the noise of no use to the trader.

C#

```
public FinVolumeAndMAPlot AddVolumeAndMAChart(
        int maperiod,
        int smoothingmode
)

public FinVolumeAndMAPlot AddVolumeAndMAChart()

public FinVolumePlot AddVolumeChart()
```

VB

```
Public Function AddVolumeAndMAChart ( _
        maperiod As Integer, _
        smoothingmode As Integer _
) As FinVolumeAndMAPlot

Public Function AddVolumeAndMAChart As  FinVolumeAndMAPlot

Public Function AddVolumeChart As  FinVolumePlot
```

If you just want the volume, without the moving average line, use:

```
public AddVolumeChart AddVolumeChart()
```

Parameters

maperiod
Type: Int32
Default: 13
period of the moving average

smoothingmode
Type: Int32
Default: FinChartConstants.MA_CALC
Specify FinChartConstants.MA_CALC for a simple moving average, and EXP_MOVING_AVERAGE_TIMEPERIOD for an exponential moving average.

Return Value

Returns the FinVolumeAndMAPlot object for the secondary chart.

Limit Values

There are no limit lines for a Volume indicator chart.

Plot Object Colors

If you want to change the colors of the lines in the Volume and MA chart, use methods found in the FinVolumeAndMAPlot class. Change the FinVolumeAndMAPlot line attributes using code similar to below.

C#

```
FinVolumeAndMAPlot vmachart = this.AddVolumeAndMAChart();
vmachart.VolumePlotAttribute.PrimaryColor = Color.Purple;
vmachart.VolumeMAPlotAttribute.LineWidth = 2;
```

VB

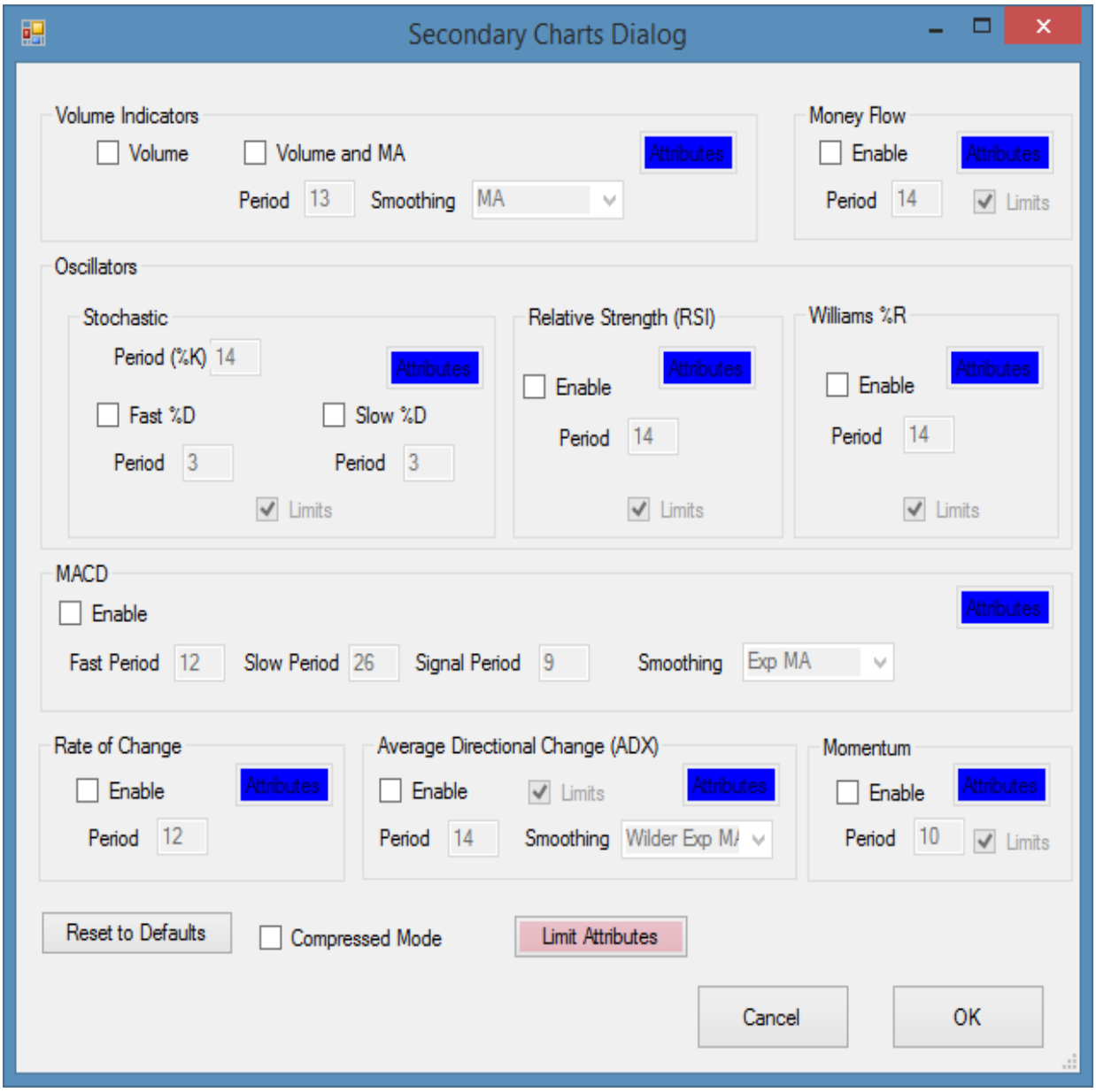
```
Dim vmachart As FinVolumeAndMAPlot = Me.AddVolumeAndMAChart()
```

```
vmachart.VolumePlotAttribute.PrimaryColor = Color.Purple
vmachart.VolumeMAPlotAttribute.LineWidth = 2
```

Secondary Chart Dialog

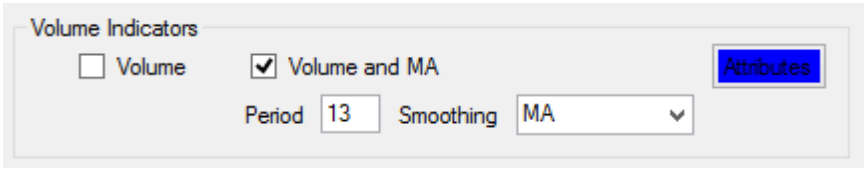
Exactly the same thing can be achieved, without programming, using the Secondary chart dialog, which is invoked using the S button in the lower right-hand corner of the Primary Chart. Start with no, or a single Secondary chart, and customize it from there.

```
this.AddPrimaryChart("TXN",  ChartObj.OHLC);

FinChartPlotBase rsichart = this.AddRSIIndicatorChart();
```



Volume Indicators

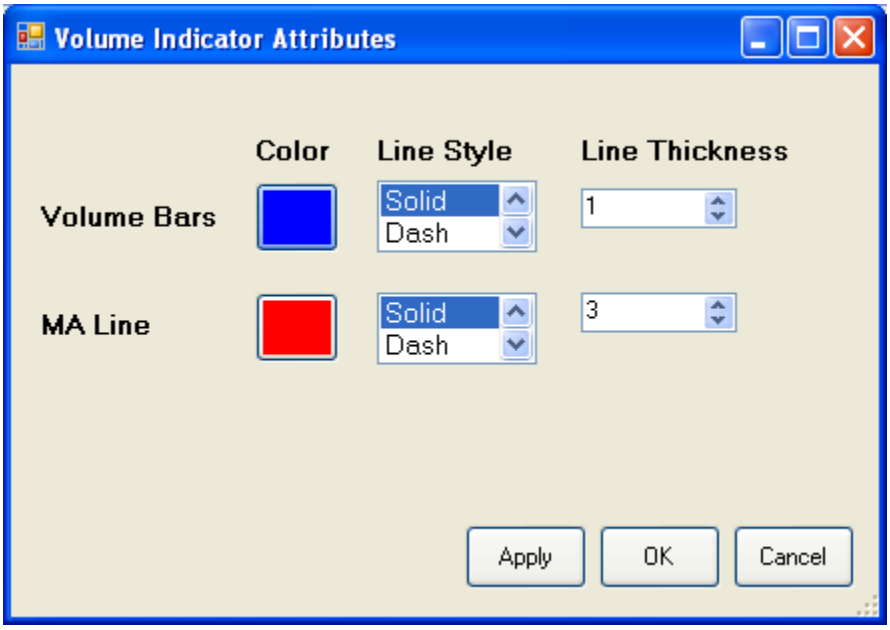
Plot a volume bar plot, or a volume bar plot with a moving average of the volume as a line plot.



Volume – Enable a Volume bar plot

Volume and MA – Enable a Volume bar plot with a moving average line plot of volume
Period– The period of the moving average
Smoothing – The moving average type: Specify Simple moving average, exponential moving average, or Wilder exponential moving average

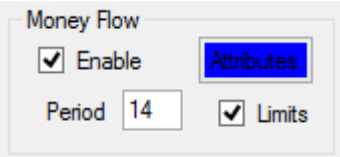
Attributes – Plot object attributes



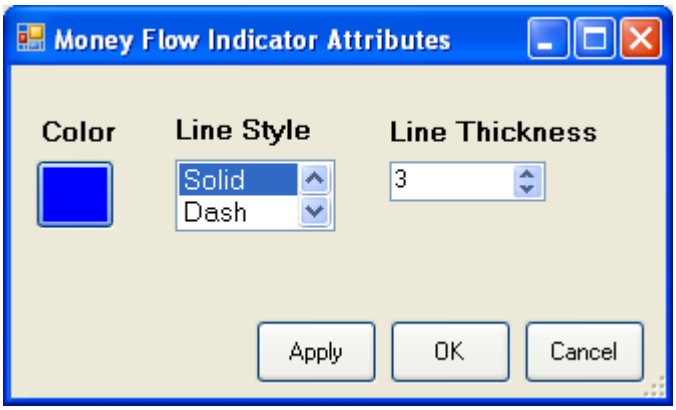
Attribute options for Volume Indicator
Volume Bars – Volume Indicator Bars
MA Line - Volume Moving Average line
Color – Primary color for the object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Money Flow

Create a Money Flow indicator chart



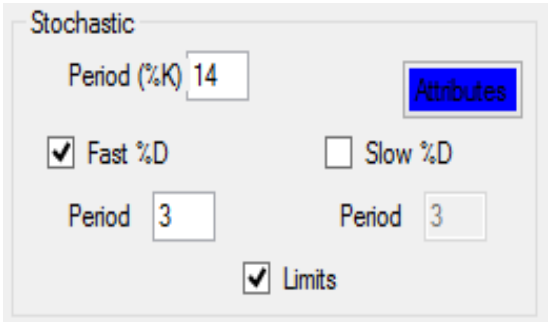
Enable – Enable the Money Flow Indicator chart
Period– The period used in the Money Flow calculation
Limits– Display limits in the chart
Attributes – Plot object attributes



Attribute options for Money Flow Indicator
Color – Primary color for the object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Stochastic

Create a Stochastic indicator chart



Period – The period used in the Stochastic %K calculation

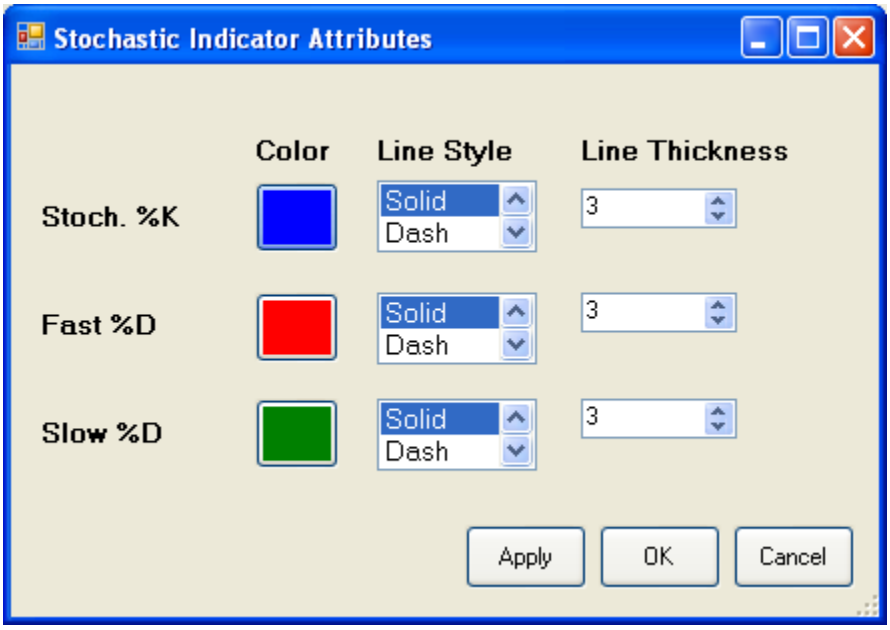
Fast (%D) – Enable the display of the Fast Stochastic line plot
Period - The period of the Fast Stochastic calculation

Slow (%D) – Enable the display of the Slow Stochastic line plot

Period - The period of the Slow Stochastic calculation

Limits– Display limits in the chart

Attributes – Plot object attributes

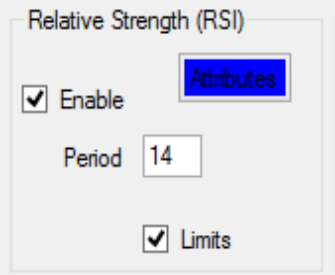


Attribute options for Stochastic Indicator

Stochastic %K – Stochastic %K line
Fast %D - Stochastic Fast %D line
Slow %D - Stochastic Slow %D line
Color – Primary color for the object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Relative Strength (RSI)

Create a RSI indicator chart



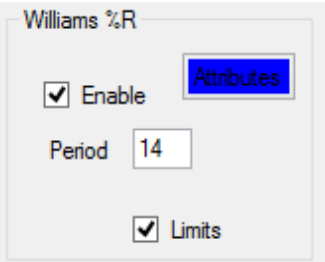
Enable – Enable the RSI Indicator chart
Period– The period used in the RSI calculation
Limits– Display limits in the chart
Attributes – Plot object attributes

Attribute options for RSI Indicator

Color – Primary color for the object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Williams (%R) Indicator

Create a Williams (%R) indicator chart



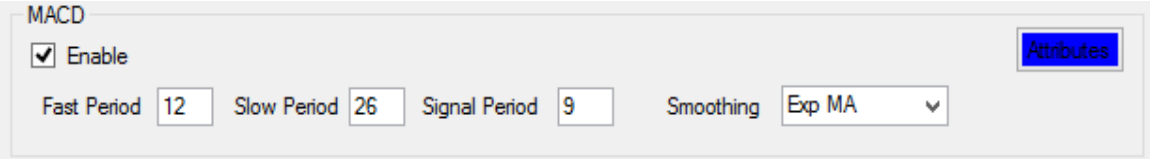
Enable – Enable the Williams (%R) indicator chart
Period– The period used in the Williams (%R) calculation
Limits– Display limits in the chart
Attributes – Plot object attributes

Attribute options for Williams (%R) Indicator

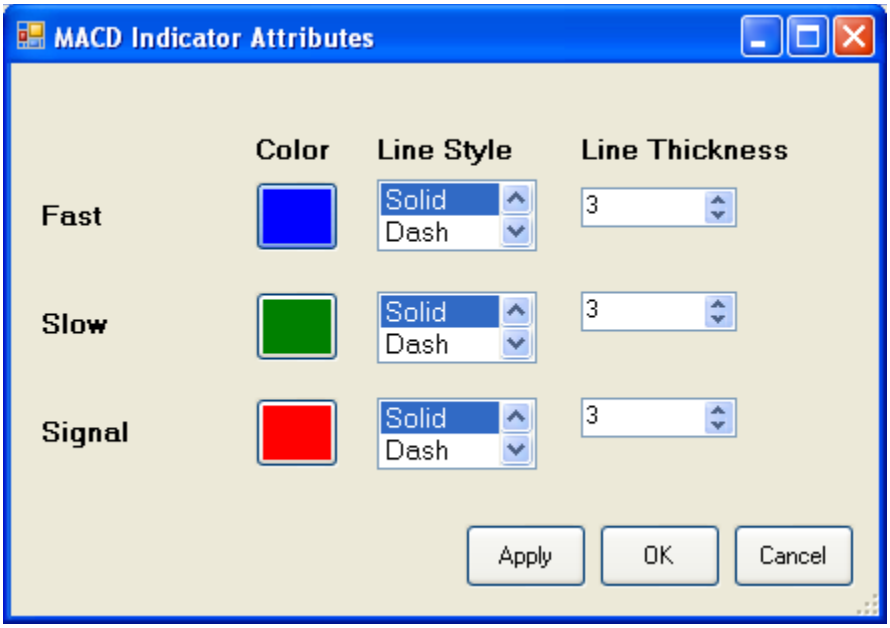
Color – Primary color for the object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

MACD

Create a MACD indicator chart



Enable – Enable the MACD indicator chart
Fast Period – The fast period used in the MACD chart
Slow Period – The slow period used in the MACD chart
Smoothing – The moving average type: Specify Simple moving average, exponential moving average, or Wilder exponential moving average
Attributes – Plot object attributes

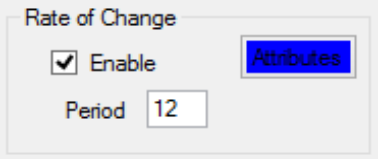


Attribute options for MACD Indicator

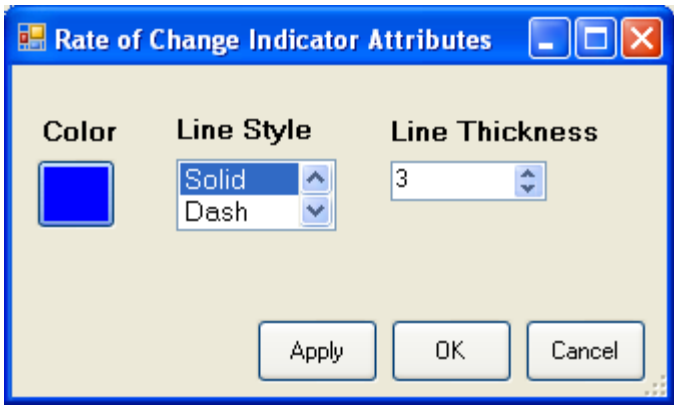
Fast MACD Line
Slow MACD Line
Signal MACD Line

Color – Primary color for the object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

7. Secondary Chart Options



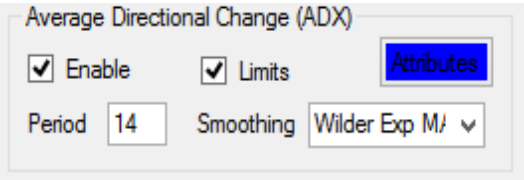
Rate of Change – Create a Rate of Change indicator chart
Enable – Enable the Rate of Change indicator chart
Period– The period used in the Rate of Change calculation
Attributes – Plot object attributes



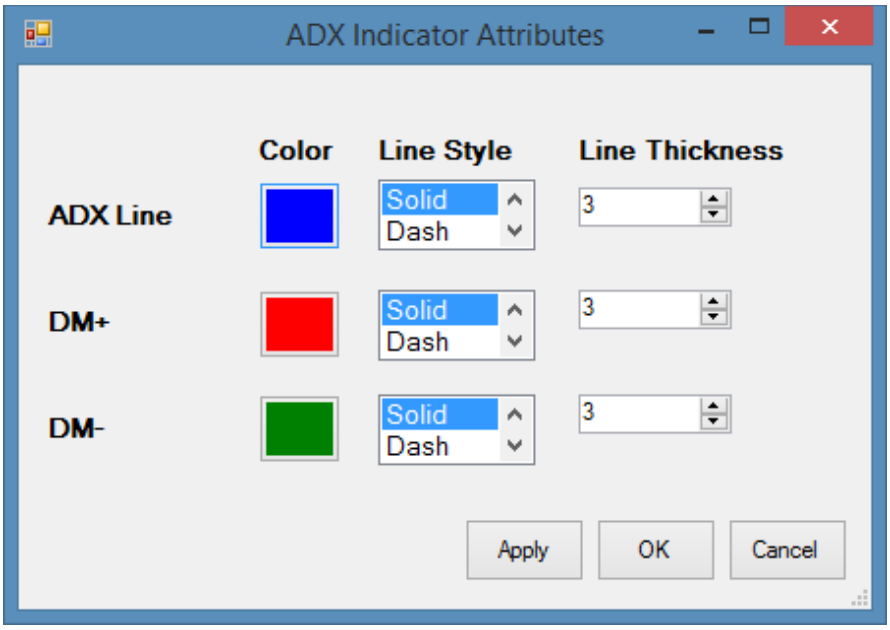
Attribute options for Rate of Change Indicator
Color – Primary color for the object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Average Directional Change (ADX)

Create a ADX indicator chart



Enable – Enable the ADX indicator chart
Period – The fast period used in the ADX chart
Smoothing – The moving average type: Specify Simple moving average, exponential moving average, or Wilder exponential moving average
Limits– Display limits in the chart
Attributes – Plot object attributes



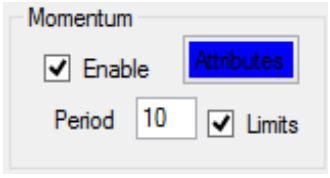
Attribute options for ADX Indicator

ADX Line
DM+ Line
DM- Line

Color – Primary color for the object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

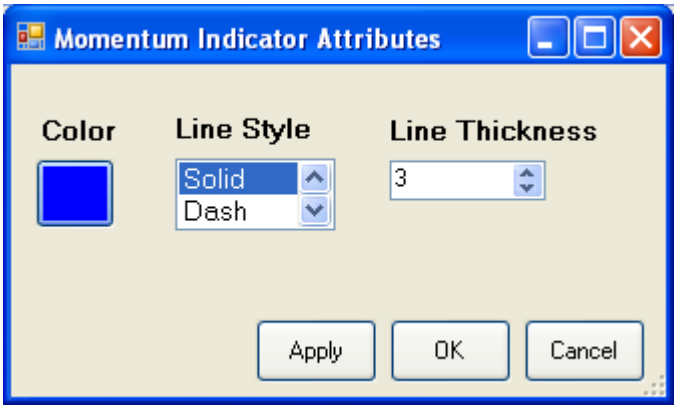
Momentum

Create a Momentum indicator chart



Enable – Enable the Momentum indicator chart
Period– The period used in the Momentum calculation
Limits– Display limits in the chart
Attributes – Plot object attributes

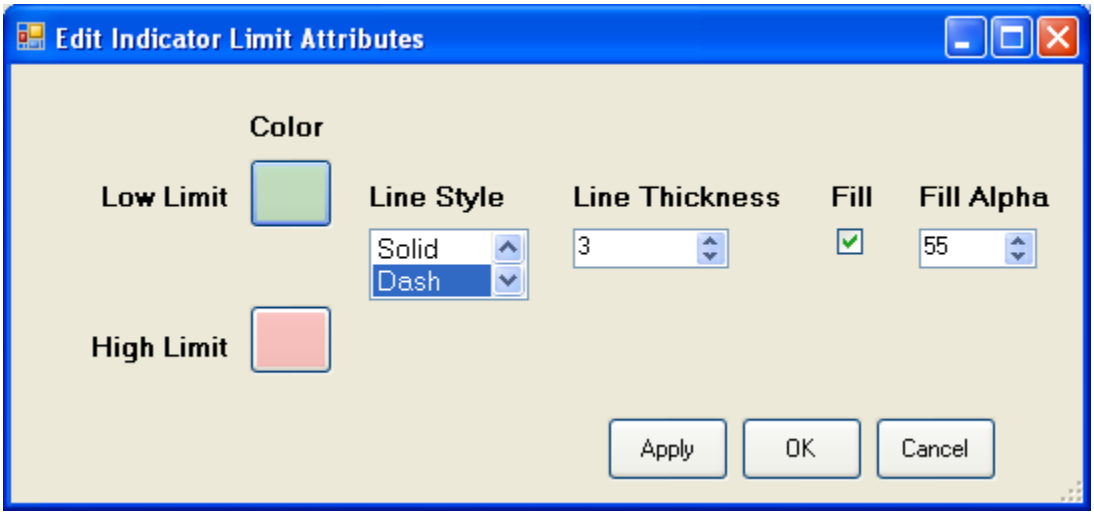
7. Secondary Chart Options



Attribute options for Momentum Indicator

Color – Primary color for the object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Limit Attributes – Limit attributes



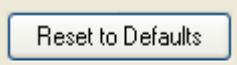
Attribute options for Limit Attributes

High Limit
Low Limit

Color – Primary line, or outline color of the plot object
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)
Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)
Fill – Check and the area between the chart edge and the limit line is filled using Color.
Fill Alpha – The transparency of the fill color – use value in the range (0..255)

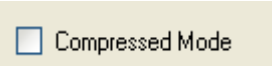
Reset to Defaults

Reset the dialog to its defaults.

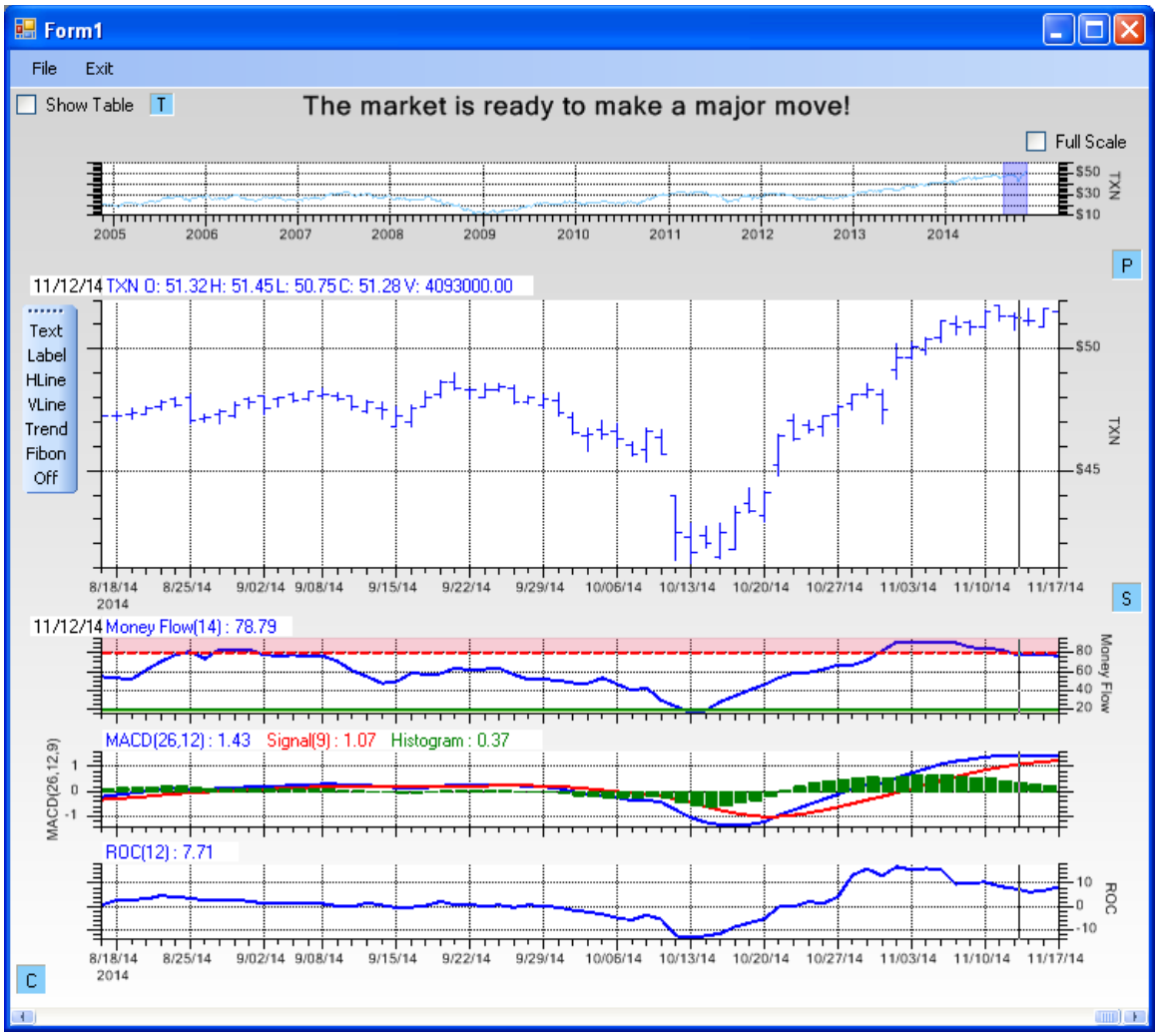


Compressed Mode

Check this and the space between adjacent Secondary indicator charts is miimized.

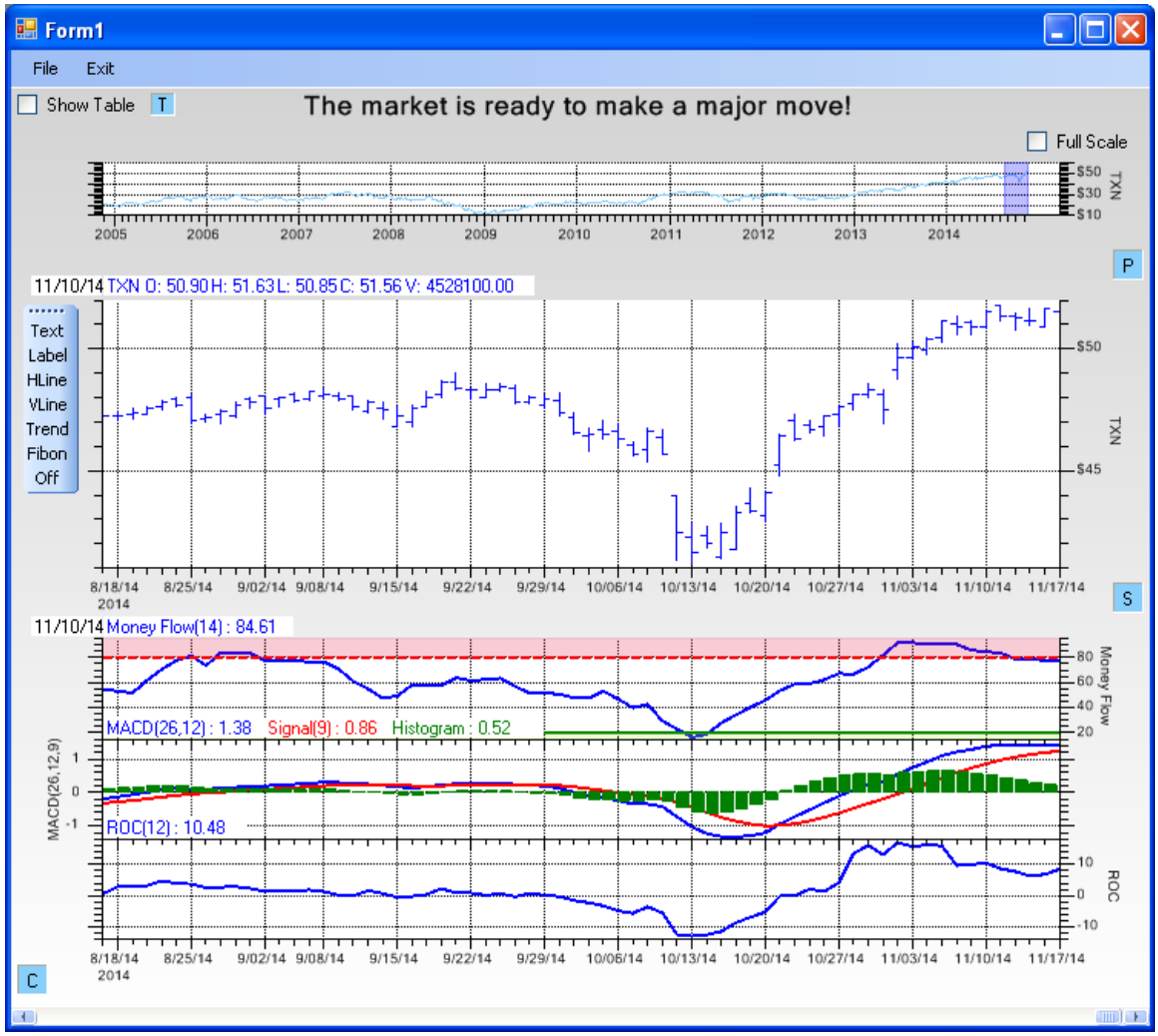


Uncompressed



7. Secondary Chart Options

Compressed



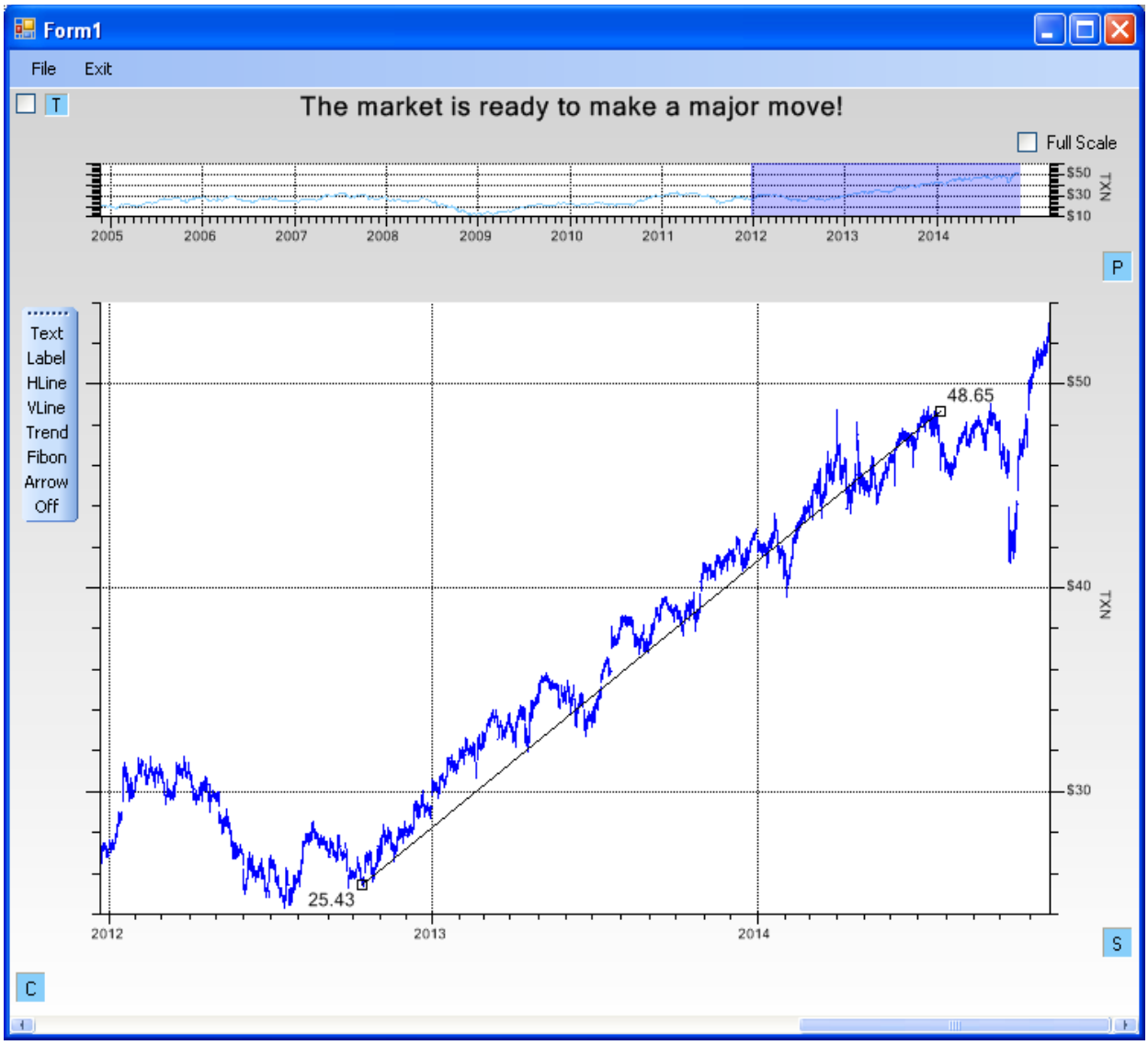
Cancel – Close dialog and cancel any changes

OK – Close dialog and apply changes to the chart.

8. Financial Chart Objects

There is a small set of technical analysis objects which can be placed in the Primary chart. These are trend lines (FinTrendLine), a financial Fibonacci object (FinFibonacciPlot), horizontal (FinHLine) and vertical (FinVLine) data markers, and two types of labels for annotations. The first label type (FinText) is positioned using normalized coordinates, and does not scroll when the chart scrolls. The second label type (FinLabel) is specified using physical coordinates and will scroll when the chart is scrolled.

Trend Line

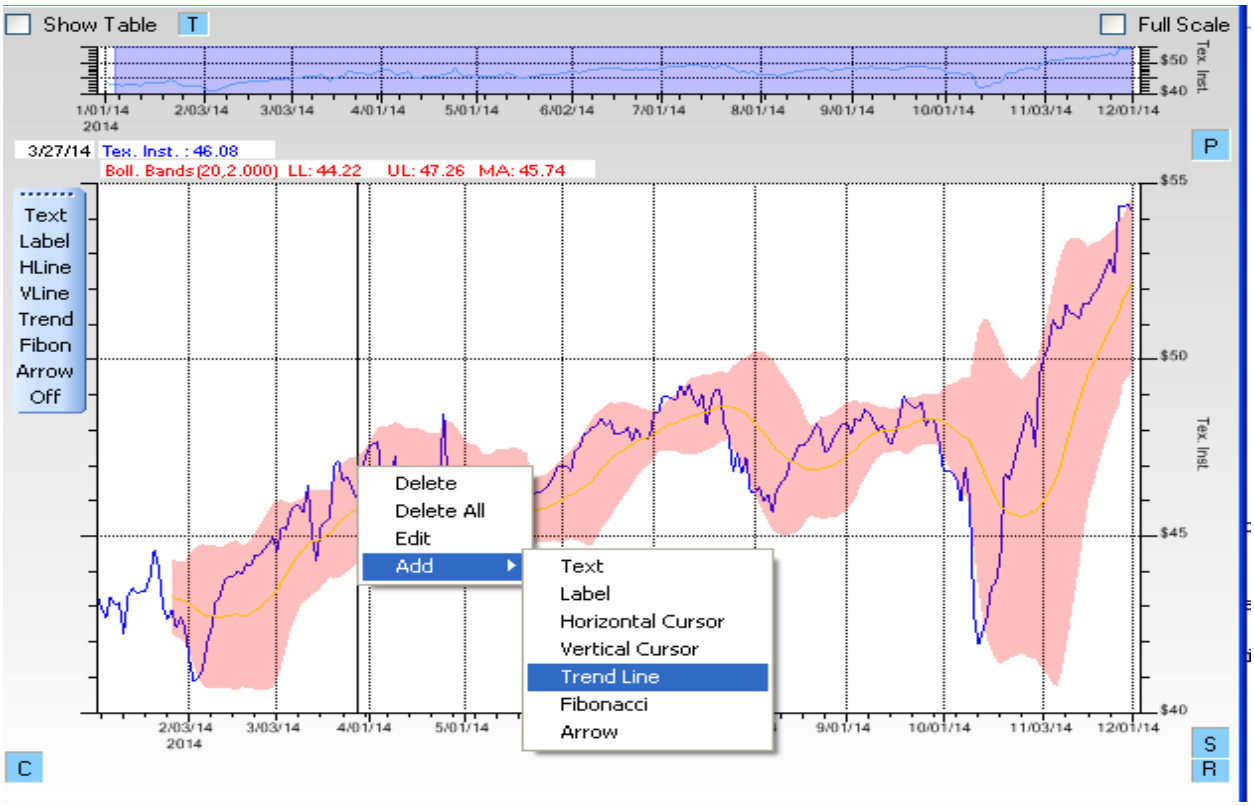


The trend line is used to mark the start and end of a trend for a stock.

Define the trend line by first selecting the Trend option from the toolstrip at the left of the graph. Then, left-click the mouse in succession at the starting and ending point (**not** a click and drag, rather two distinct

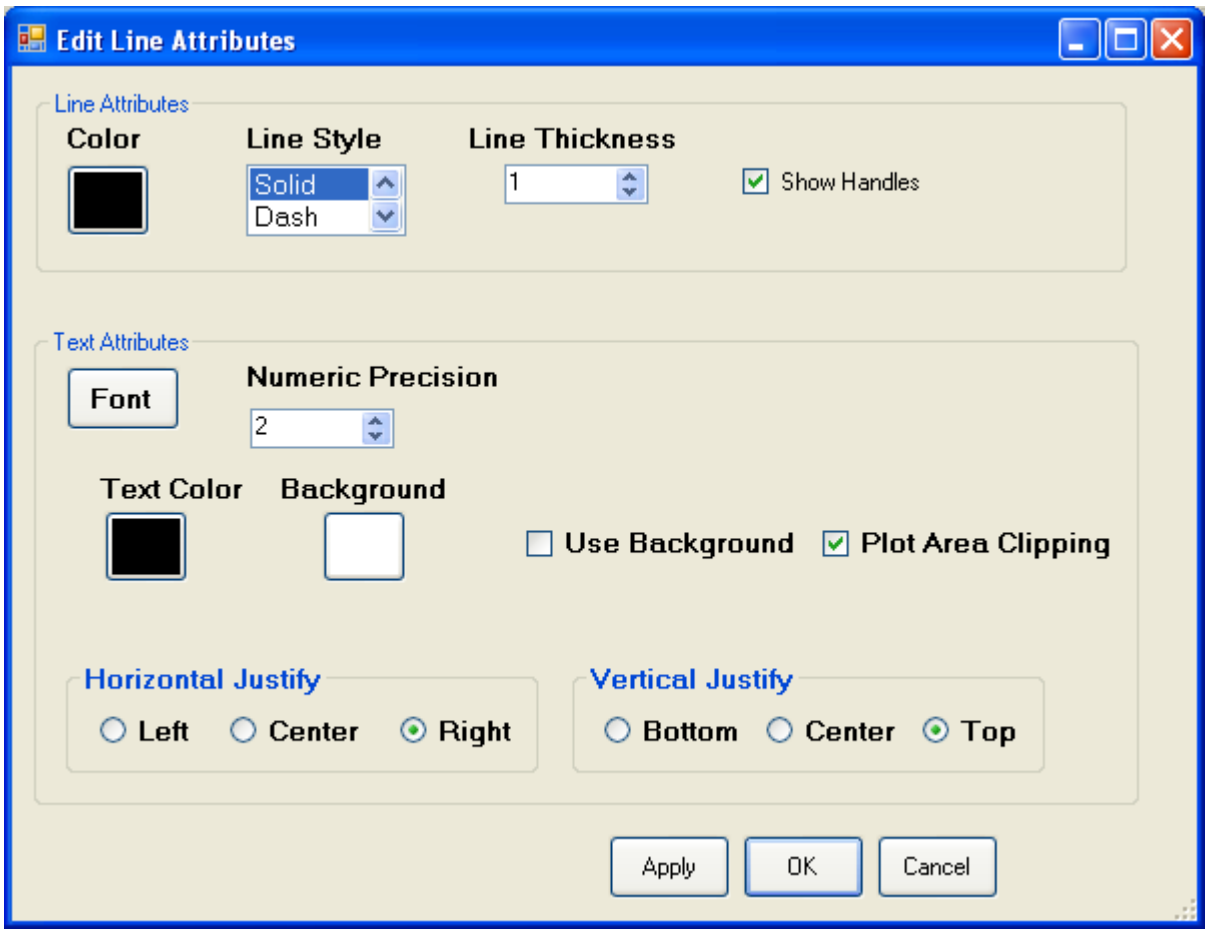
single clicks). There is no need to be exact, since you can adjust the trend line after it is created. The trend line will connect the two points where you clicked.

You can also define a trend line by right clicking on the chart and selecting **Add | Trend Line** from the pop-up menus. Then left click for the starting, and left click for the ending, positions of the trend line.



Once placed, you can still adjust the position of the line with great precision.. You can refine the position, moving it by left-click-dragging the center of the line. The slope of the line can be changed by left-click-dragging on either endpoint of the trend line. The price values of the trend line endpoints are also displayed.

You can edit the trend line properties by left-double-clicking on it. That will invoke the trend line edit dialog, seen below.



Attribute options for the Trend Line Dialog

Line Attributes

Color – Primary color for the object

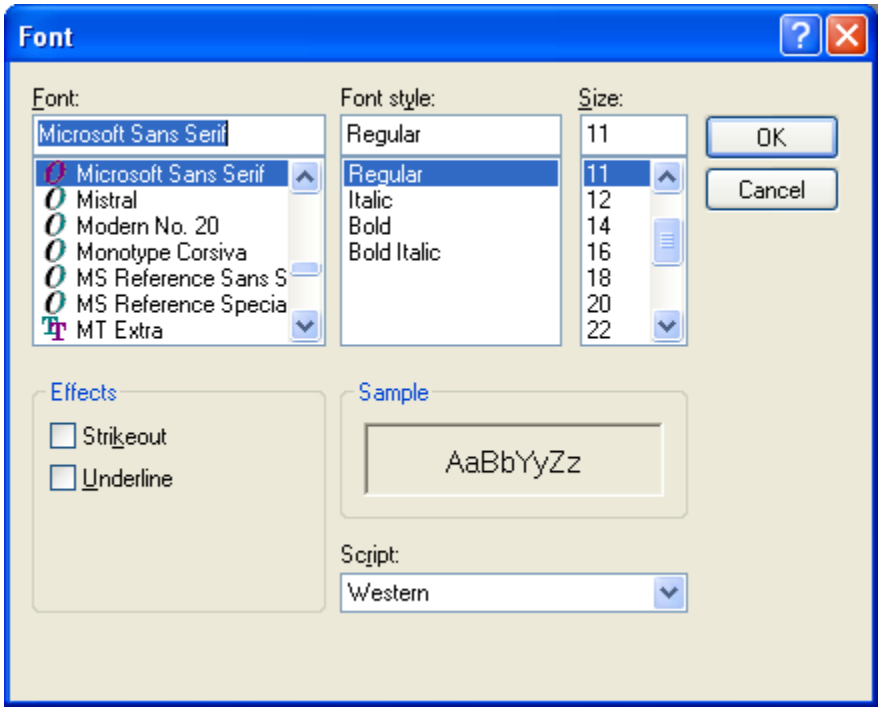
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)

Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Show Handles – Check and the grab handles of the trend line are displayed at the endpoints.

Text Attributes

Font – Edit the text Font and Font Size



Numeric precision – Set the numeric precision of the text labels.

Text Color – Select the color of the text

Background – Select the background color of the text

Use Background – Check this box if you want the background rectangle of the text to overwrite what is underneath.

Plot Area Clipping – Check this box if you want the text to clip to the plotting area (the area bounded by the axes)

Horizontal Justify – Sets horizontal text justification of the trend line start label. The justification of the trend line end label is set to the opposite value.

Vertical Justify – Sets vertical text justification of the trend line start label. The justification of the trend line end label is set to the opposite value.

You can add a trend line programmatically using the FinChartView AddFinTrendLineToPrimaryChart method. **You MUST add it after the FinChartView.BuildChart** call**.**

Prototypes

C#

```
public FinTrendLine AddFinTrendLineToPrimaryChart(
        bool update,
        ChartEvent ev1,
```

```
        ChartEvent ev2
)

Public FinTrendLine AddFinTrendLineToPrimaryChart(
        bool update
)
```

VB

```
Public Function AddFinTrendLineToPrimaryChart ( _
        update As Boolean, _
        ev1 As ChartEvent, _
        ev2 As ChartEvent _
) As FinTrendLine

Public Function AddFinTrendLineToPrimaryChart ( _
        update As Boolean _
) As FinTrendLine
```

where AddFinTrendLineToPrimaryChart has the following characteristics.

Parameters

update
Type: Boolean
Force an immediate update of the chart, rebuilding in the process.

ev1
Type: ChartEvent
The starting position of the trend line.

ev2
Type: ChartEvent
The ending position of the trend line.

Return Value
Returns the FinTrendLine object created.

Examples

In this case, a default trend line is created in the middle of the current display, and it is up to the user to reposition the endpoints of the trend line using the mouse.

C#

```
this.AddPrimaryChart("TXN", ChartObj.OHLC);


this.BuildChart();

this.AddFinTrendLineToPrimaryChart(true);
```

VB

```
Me.AddPrimaryChart("TXN", ChartObj.OHLC)

Me.BuildChart();

Me.AddFinTrendLineToPrimaryChart(True)
```

In this case, the trendline is defined using OHLC events. The user can still reposition the trend line once it is displayed on the chart.

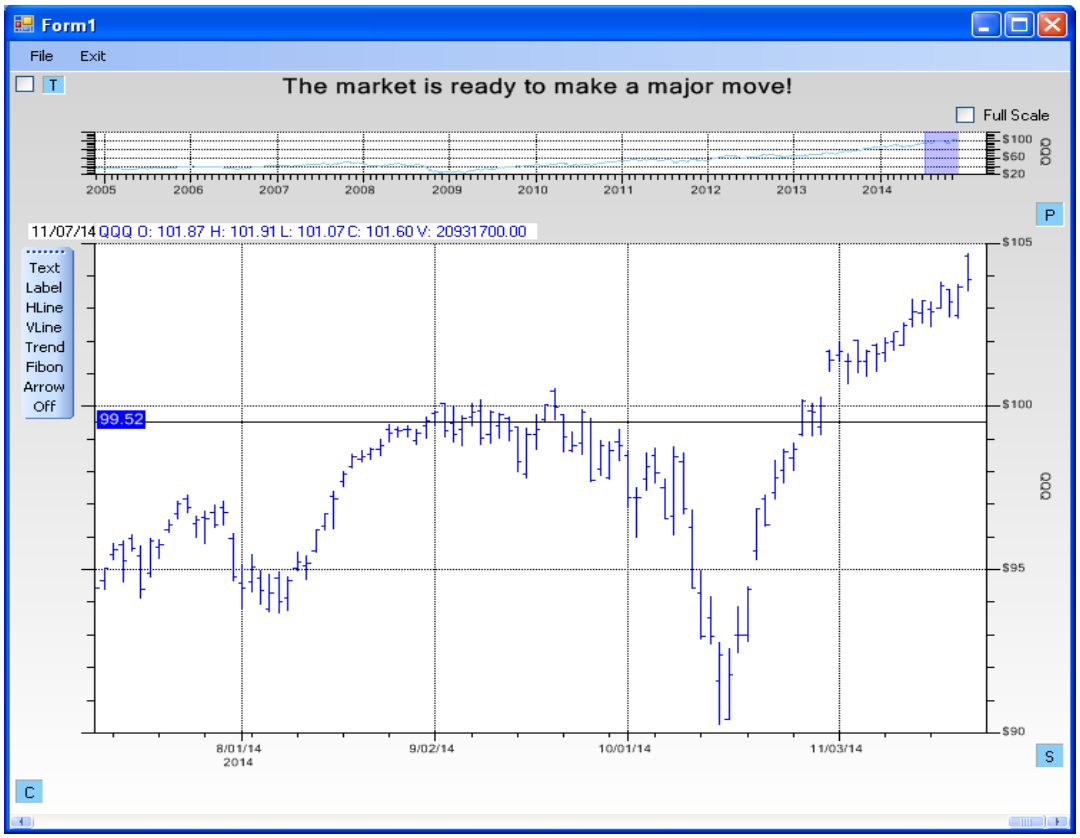
C#

```
EventGroupDataset rawStockData = this.GetCurrentTickerData();

ChartEvent tlev1 = rawStockData.GetEvent(30);
ChartEvent tlev2 = rawStockData.GetEvent(70);
FinTrendLine fintrendline1 = this.AddFinTrendLineToPrimaryChart(true, tlev1, tlev2);
```

VB

```
Dim rawStockData As EventGroupDataset = Me.GetCurrentTickerData()

Dim tlev1 As ChartEvent = rawStockData.GetEvent(30)
Dim tlev2 As ChartEvent = rawStockData.GetEvent(70)
Dim fintrendline1 As FinTrendLine = Me.AddFinTrendLineToPrimaryChart(True, tlev1, tlev2)
```

Horizontal Data Marker

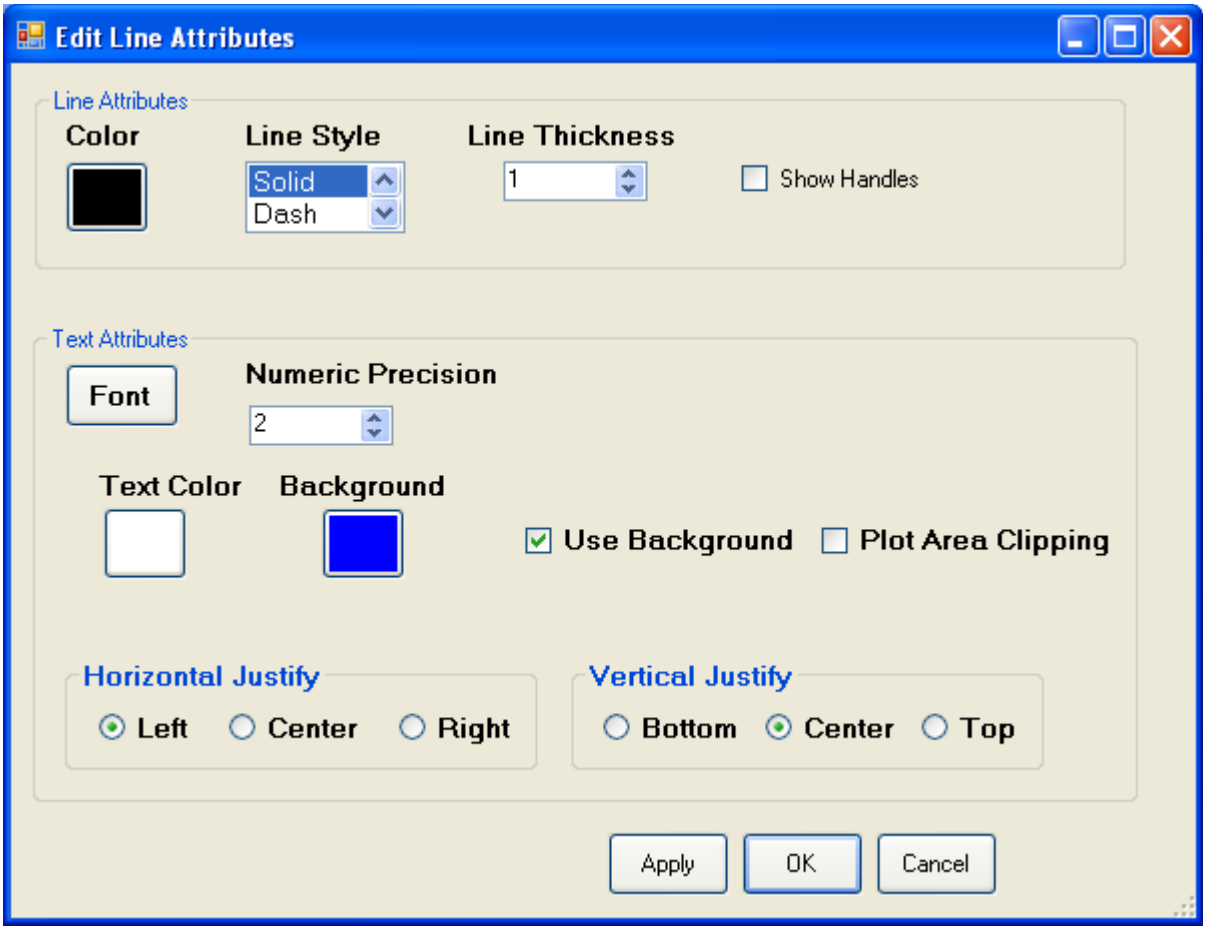


The horizontal data marker can be used to mark support and resistance levels.

Define the horizontal data marker line by first selecting the HLine option from the toolstrip at the left of the graph. Then, left-click the mouse at the vertical position where you want the horizontal cursor. There is no need to be exact, since you can adjust the cursor position after it is created.

You can also define a horizontal cursor by right clicking on the chart and selecting **Add | Horizontal Cursor** from the pop-up menu. Then left click for the position of the horizontal cursor.

Once placed, you can still adjust the position of the line with great precision.. You can refine the position, moving it by left-click-dragging any part of the line. The price value of the horizontal cursor is displayed on the left.



You can edit the horizontal cursor properties by left-double-clicking on it. That will invoke the horizontal cursor edit dialog, seen below.

Attribute options for the Horizontal Cursor Dialog

Line Attributes

Color – Primary color for the object

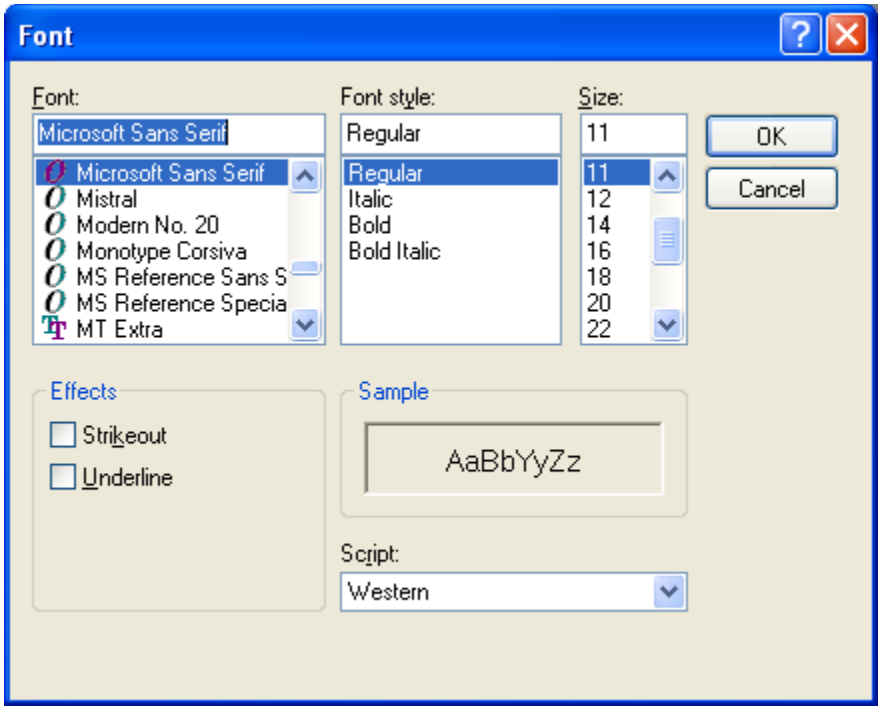
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)

Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Show Handles – Check and the grab handles of the trend line are displayed at the endpoints.

Text Attributes

Font – Edit the text Font and Font Size



Numeric precision – Set the numeric precision of the text labels.

Text Color – Select the color of the text

Background – Select the background color of the text

Use Background – Check this box if you want the background rectangle of the text to overwrite what is underneath.

Plot Area Clipping – Check this box if you want the text to clip to the plotting area (the area bounded by the axes)

Horizontal Justify – Sets horizontal text justification of the trend line start label. The justification of the trend line end label is set to the opposite value.

Vertical Justify – Sets vertical text justification of the trend line start label. The justification of the trend line end label is set to the opposite value.

You can add a horizontal line programmatically using the FinChartView AddFinHLineToPrimaryChart method. **You MUST add it after the FinChartView.BuildChart call.**

Prototypes

C#

```
public FinHLine AddFinHLineToPrimaryChart(
        bool update,
```

```
        ChartEvent ev1
)

public FinHLine AddFinHLineToPrimaryChart(
        bool update
)
```

VB

```
Public Function AddFinHLineToPrimaryChart ( _
        update As Boolean, _
        ev1 As ChartEvent _
) As FinHLine

Public Function AddFinHLineToPrimaryChart ( _
        update As Boolean _
) As FinHLine
```

where AddFinHLineToPrimaryChart has the following characteristics.

Parameters

update
Type: Boolean
Force an immediate update of the chart, rebuilding in the process.

ev1
Type: ChartEvent
The starting position of the cursor.

Return Value
Returns the FinHLabel object created.

Examples

In this example, a horizontal cursor is created in the middle of the current display, and it is up to the user to reposition it using the mouse.

C#

```
this.AddPrimaryChart("TXN", ChartObj.OHLC);

this.BuildChart();

this.AddFinHLineToPrimaryChart(true);
```

VB

```
Me.AddPrimaryChart("TXN",  ChartObj.OHLC)

Me.BuildChart()

Me.AddFinHLineToPrimaryChart(True)
```

In this example, a horizontal cursor is created at a position defined by an OHLC stock value. The user can still reposition it using the mouse.

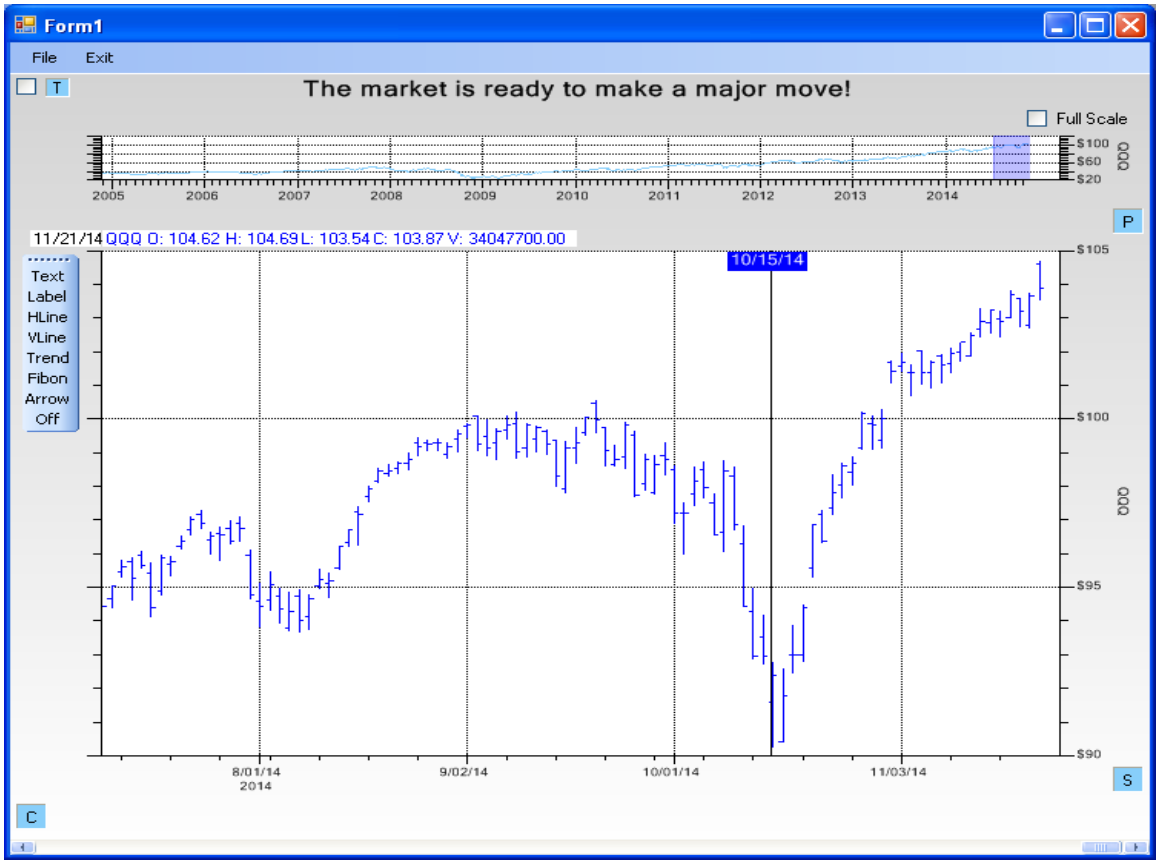
C#

```
this.AddPrimaryChart("TXN",  ChartObj.OHLC);

this.BuildChart();

EventGroupDataset rawStockData = this.GetCurrentTickerData();
ChartEvent hlev1 = rawStockData.GetEvent(60);
FinHLine finhline1 = this.AddFinHLineToPrimaryChart(true, hlev1);
```

VB

```
Me.AddPrimaryChart("TXN",  ChartObj.OHLC)

Me.BuildChart()

Dim rawStockData As EventGroupDataset = Me.GetCurrentTickerData()
Dim hlev1 As ChartEvent = rawStockData.GetEvent(60)
Dim finhline1 As FinHLine = Me.AddFinHLineToPrimaryChart(True, hlev1)
```

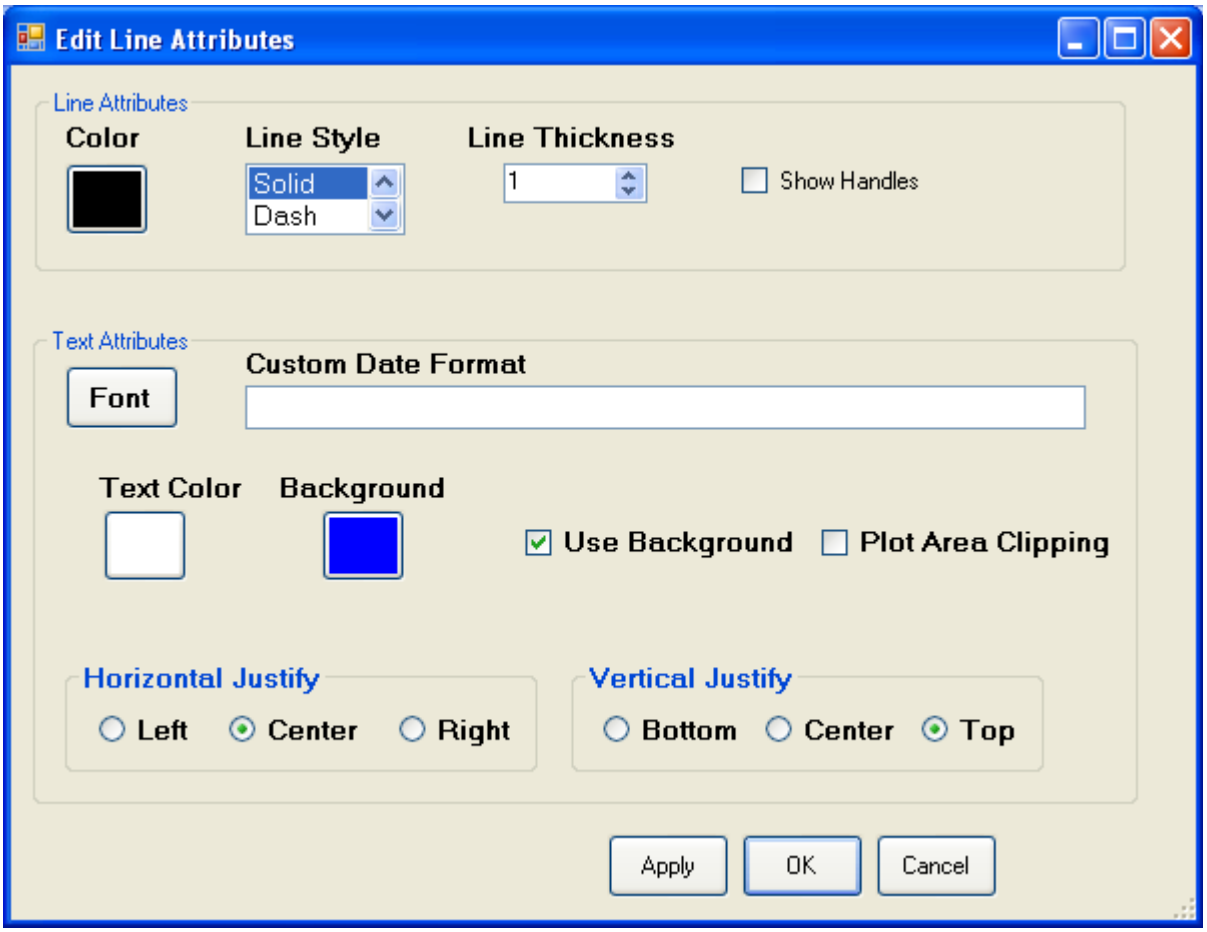
Vertical Data Marker



Define the vertical data cursor line by first selecting the VLine option from the toolstrip at the left of the graph. Then, left-click the mouse at the horizontal position where you want the vertical cursor. There is no need to be exact, since you can adjust the cursor position after it is created.

You can also define a vertical cursor by right clicking on the chart and selecting **Add | Vertical Cursor** from the pop-up menu. Then left click for the position of the vertical cursor.

Once placed, you can still adjust the position of the line with great precision.. You can refine the position, moving it by left-click-dragging any part of the line. The price value of the horizontal cursor is displayed on the left.



You can edit the horizontal cursor properties by left-double-clicking on it. That will invoke the horizontal cursor edit dialog, seen below.

Attribute options for the Vertical Cursor Dialog

Line Attributes

Color – Primary color for the object

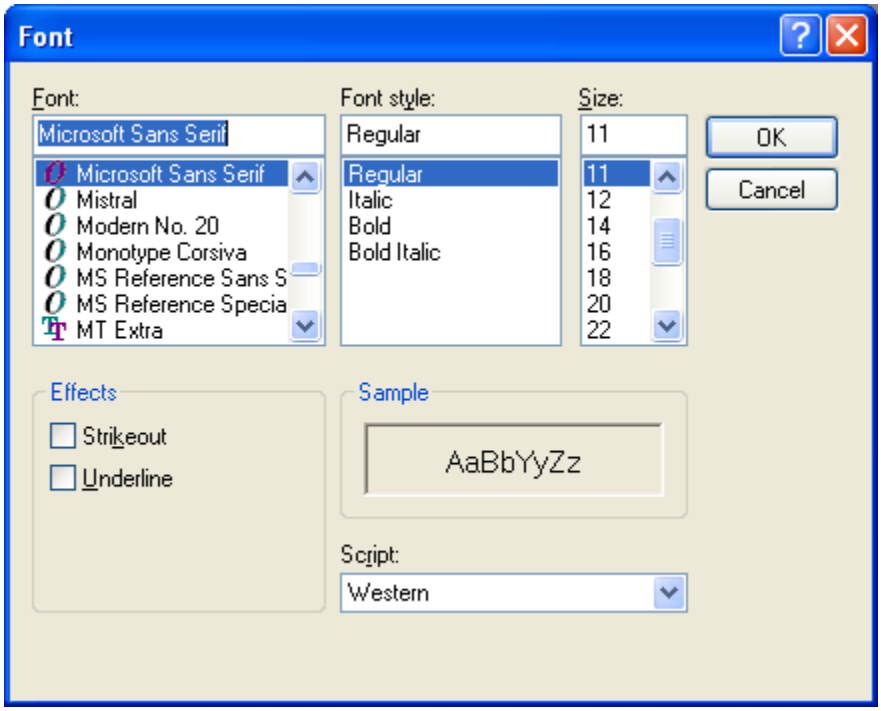
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)

Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Show Handles – Check and the grab handles of the trend line are displayed at the endpoints.

Text Attributes

Font – Edit the text Font and Font Size



Custom Date Format– Set a custom date format using the standard .Net date formatting rules. The default is mm/dd/yy

Text Color – Select the color of the text

Background – Select the background color of the text

Use Background – Check this box if you want the background rectangle of the text to overwrite what is underneath.

Plot Area Clipping – Check this box if you want the text to clip to the plotting area (the area bounded by the axes)

Horizontal Justify – Sets horizontal text justification of the trend line start label. The justification of the trend line end label is set to the opposite value.

Vertical Justify – Sets vertical text justification of the trend line start label. The justification of the trend line end label is set to the opposite value.

You can add a vertical line programmatically using the FinChartView AddFinVLineToPrimaryChart method. **You MUST add it after the FinChartView.BuildChart call.**

Prototypes

C#

```
public FinVLine AddFinVLineToPrimaryChart(
        bool update,
        ChartEvent ev1
)
public FinVLine AddFinVLineToPrimaryChart(
```

```
        bool update
)
```

VB

```
Public Function AddFinVLineToPrimaryChart ( _
        update As Boolean, _
        ev1 As ChartEvent _
) As FinVLine

Public Function AddFinVLineToPrimaryChart ( _
        update As Boolean _
) As FinVLine
```

where AddFinHLineToPrimaryChart has the following characteristics.

Parameters

update
Type: Boolean
Force an immediate update of the chart, rebuilding in the process.

ev1
Type: ChartEvent
The starting position of the cursor.

Return Value
Returns the FinVLabel object created.

Examples

In this example, a vertical cursor is created in the middle of the current display, and it is up to the user to reposition it using the mouse.

C#

```
this.AddPrimaryChart("TXN",  ChartObj.OHLC);

this.BuildChart();

this.AddFinVLineToPrimaryChart(true);
```

VB

```
Me.AddPrimaryChart("TXN",  ChartObj.OHLC)

Me.BuildChart()

Me.AddFinVLineToPrimaryChart(True)
```

In this example, a vertical cursor is created at a position defined by an OHLC stock value. The user can still reposition it using the mouse.

C#

```
this.AddPrimaryChart("TXN",  ChartObj.OHLC);

this.BuildChart();

EventGroupDataset rawStockData = this.GetCurrentTickerData();
ChartEvent vlev1 = rawStockData.GetEvent(60);
FinVLine finvline1 = this.AddFinVLineToPrimaryChart(true, vlev1);
```

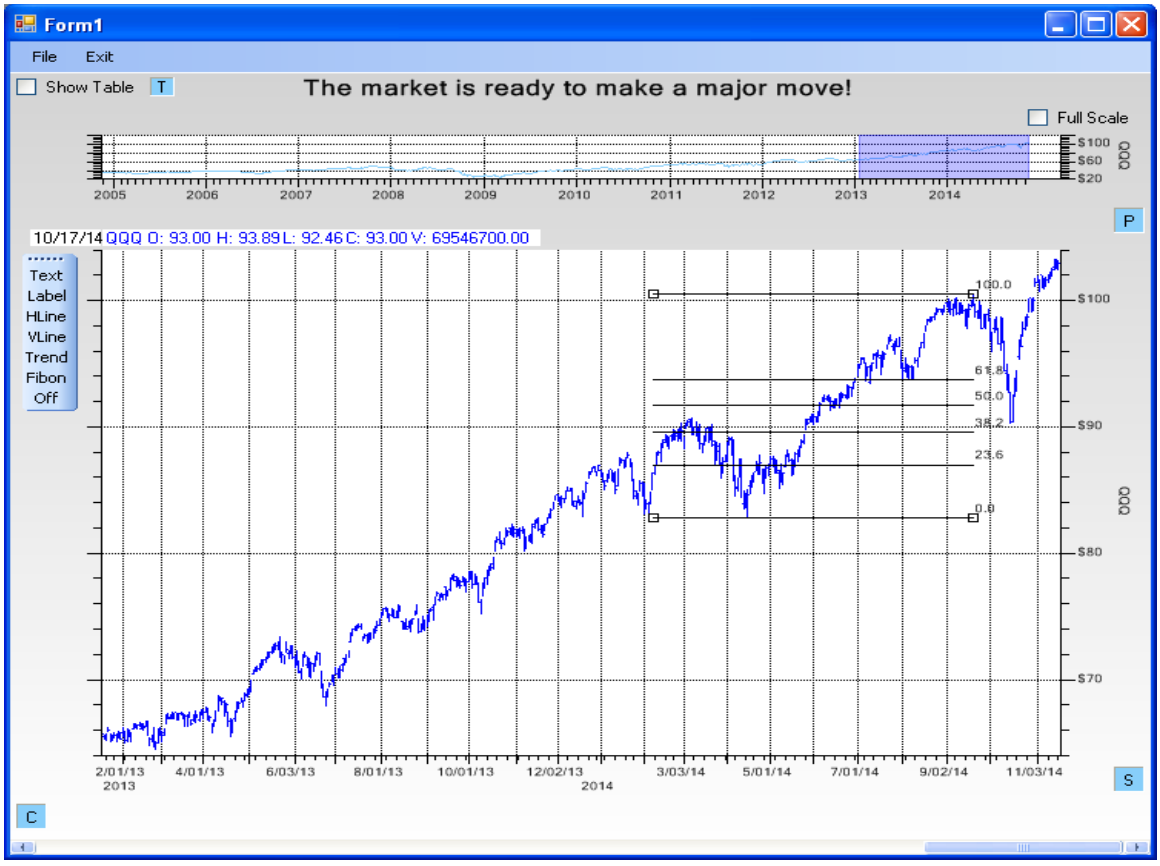
VB

```
Me.AddPrimaryChart("TXN",  ChartObj.OHLC)

Me.BuildChart()

Dim rawStockData As EventGroupDataset = Me.GetCurrentTickerData()
Dim vlev1 As ChartEvent = rawStockData.GetEvent(60)
Dim finvline1 As FinVLine = Me.AddFinVLineToPrimaryChart(True, vlev1)
```

Fibonacci Overlay

Define the Fibonacci overlay by first selecting the Fibonacci option from the toolstrip at the left of the graph. Then, left-click the mouse in succession at the lower-left and upper-right corners of the desired Fibonacci overlay (**not** a click and drag, rather two distinct single clicks). There is no need to be exact, since you can adjust the Fibonacci overlay after it is created.

You can also define a Fibonacci overlay by right clicking on the chart and selecting **Add | Fibonacci** from the pop-up menus. Then left click for the lower-left corner of the Fibonacci overlay, and left click again for the upper-right corner of the overlay.



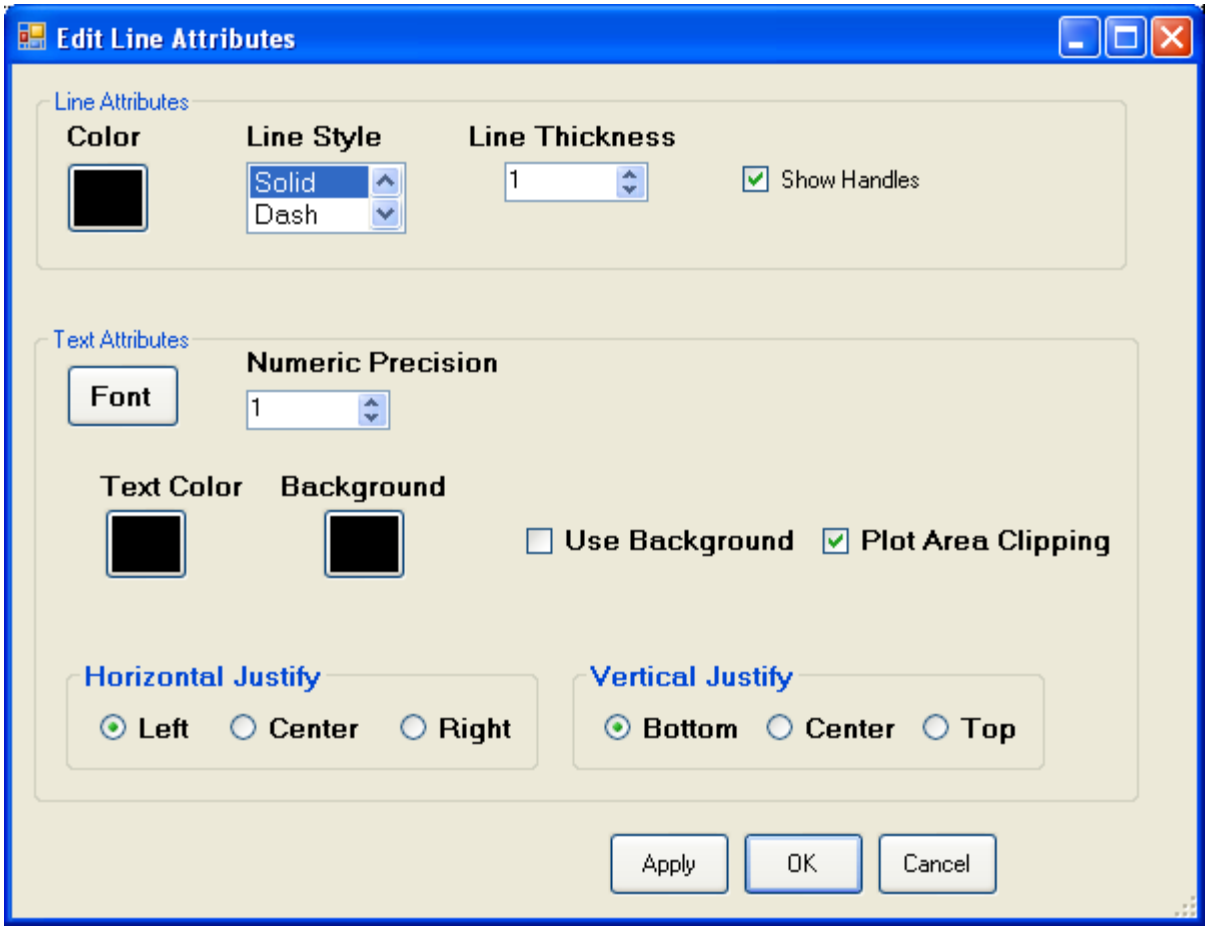
The 0 to 100% section of the Fibonacci overlay will be drawn using the first and second mouse clicks. It defines the 0% level at the first click, and the 100% level at the second click. If the first click is higher than the second click, then the Fibonacci overlay will be inverted, with the 100% level at the top, and the 0% level at the bottom. Also, there are two Fibonacci levels above 100%, which are 161.8% and 261.8%. They will be displayed if they are within

the plot area window, otherwise they will be clipped out of the chart.

Once placed, you can still adjust the position of the Fibonacci overlay with great precision.. You can refine the position, moving it by left-click-dragging anywhere in the rectangle defined by the Fibonacci overlay. The width and height of the Fibonacci overlay can be changed by left-click-dragging on any of the small grab rectangle at the 0.0% and 100% locations on the overlay.

A Fibonacci overlay is used in some types of technical analysis wave theory to identify support and resistance levels for a stock. The values on the right of the horizontal lines in the example above (0, 23.6, 38.2, 50.0, 61.8, 100, 161.8 and 261.8) are represent percentage values and are calculated using the famous Fibonacci sequence (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233).

You can edit the trend line properties by left-double-clicking on it. That will invoke the Fibonacci overlay edit dialog, seen below.



Attribute options for the Fibonacci Dialog

Line Attributes

Color – Primary color for the object

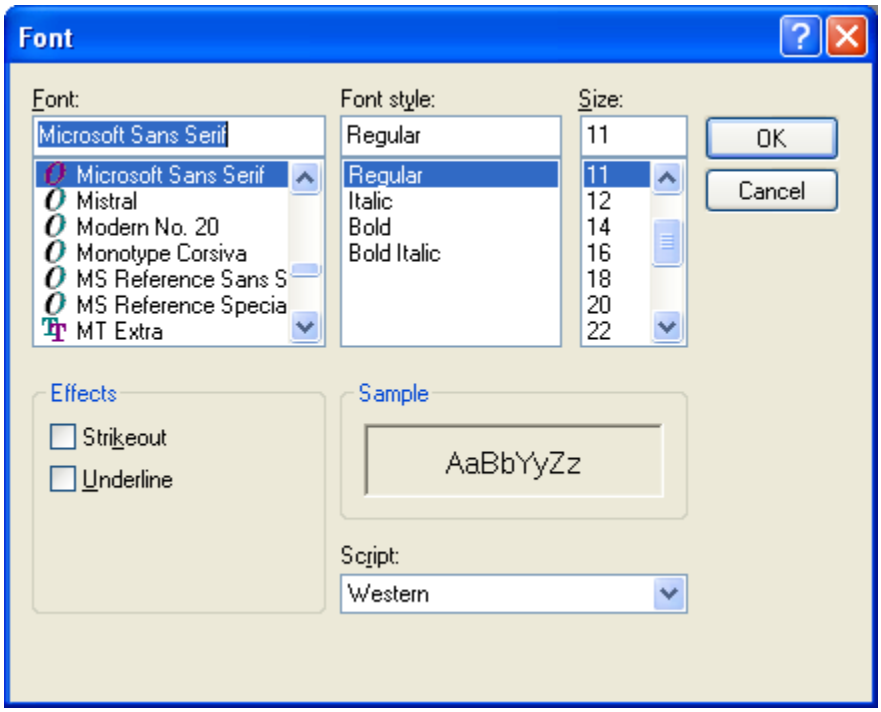
Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)

Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

Show Handles – Check and the grab handles of the trend line are displayed at the 0% and 100% levels of the Fibonacci chart..

Text Attributes

Font – Edit the text Font and Font Size



Numeric precision – Set the numeric precision of the text labels.

Text Color – Select the color of the text

Background – Select the background color of the text

Use Background – Check this box if you want the background rectangle of the text to overwrite what is underneath.

Plot Area Clipping – Check this box if you want the text to clip to the plotting area (the area bounded by the axes)

Horizontal Justify – Sets horizontal text justification of the trend line start label. The justification of the trend line end label is set to the opposite value.

Vertical Justify – Sets vertical text justification of the trend line start label. The justification of the trend line end label is set to the opposite value.

You can add a Fibonacci rectangle programmatically using the FinChartView AddFinFibonacciPlotToPrimaryChart method. **You MUST add it after the FinChartView.BuildChart call.**

Prototypes

C#

```
public FinFibonacciPlot AddFinFibonacciPlotToPrimaryChart(
        bool update,
        ChartEvent ev1,
        ChartEvent ev2
)
public FinFibonacciPlot AddFinFibonacciPlotToPrimaryChart(
        bool update
)
```

VB

```
Public Function AddFinFibonacciPlotToPrimaryChart ( _
        update As Boolean _
) As FinFibonacciPlot

Public Function AddFinFibonacciPlotToPrimaryChart ( _
        update As Boolean, _
        ev1 As ChartEvent, _
        ev2 As ChartEvent _
) As FinFibonacciPlot
```

Parameters

update
Type: Boolean
Force an immediate update of the chart, rebuilding in the process.

ev1
Type: ChartEvent
The lower-left corner of the Finbonacci rectangle.

ev2
Type: ChartEvent
The upper-right corner of the Finbonacci rectangle.

Return Value
Returns the FinFibonacciPlot object created.

Examples

In this case, a Fibonacci plot is created in the middle of the current display, and it is up to the user to reposition it using the mouse.

C#

```
this.BuildChart();

this.AddFinFibonacciPlotToPrimaryChart(true);
```

VB

```
Me.AddPrimaryChart("TXN",  ChartObj.OHLC)

Me.BuildChart();
```

```
Me.AddFinFibonacciPlotToPrimaryChart(True)
```

In this case, the Fibonacci plot is defined using OHLC events. The user can still reposition it using the mouse .

C#

```
ChartEvent fibev1 = rawStockData.GetEvent(120);
FinFibonacciPlot fibplot = this.AddFinFibonacciPlotToPrimaryChart(true, fibev1, fibev2);
```

VB

```
Dim rawStockData As EventGroupDataset = Me.GetCurrentTickerData()

Dim fibev1 As ChartEvent = rawStockData.GetEvent(120)
Dim fibev2 As ChartEvent = rawStockData.GetEvent(100)
Dim fibplot As FinFibonacciPlot =
       Me.AddFinFibonacciPlotToPrimaryChart(True, fibev1, fibev2)
```

Labels for Annotation

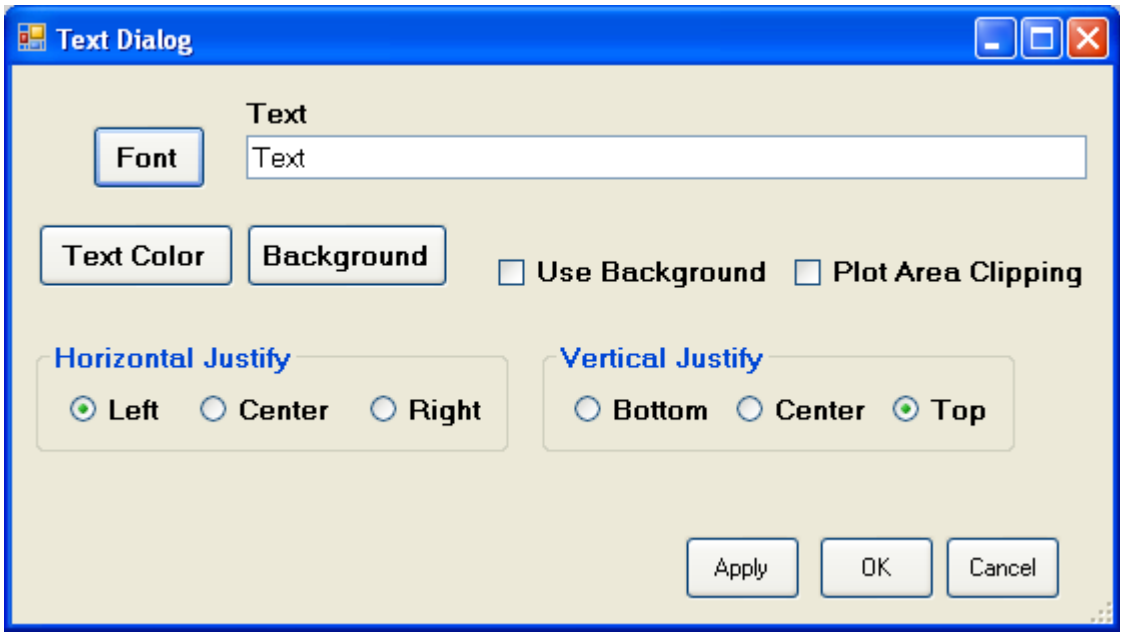
In this example, the FinText object provides a bold title for the chart.

There are three types of objects you can use for annotations. The first uses the FinText class. It is placed in the chart using Normalized graph coordinates. Because of this, it does not scroll, or change position, when the graph y-scale is changed, or when the graph is panned or zoomed. Because Normalized Graph coordinates do not change even if the graphs physical coordinate system changes. The other two are the FinLabel and FinArrow objects, which are placed using physical coordinates and do scroll with the plot.

FinText

First, select the Text option from the toolstrip on the left of the chart. Next, select a position by left clicking on the chart at the xy-position where you want the text to appear..The following dialog will appear.

.



Font – Edit the text Font and Font Size

Text – The text string to display as a label

Text Color – Select the color of the text

Background – Select the background color of the text

Use Background – Check this box if you want the background rectangle of the text to overwrite what is underneath.

Plot Area Clipping – Check this box if you want the text to clip to the plotting area (the area bounded by the axes)

Horizontal Justify – Sets horizontal text justification of the numeric label.

Vertical Justify – Sets vertical text justification of the numeric label.

Set the font you want to use, and the text string and press OK. The text string will appear at the selected xy-position.

You can also define a text object by right-clicking the mouse on the chart. When you right-click, hold the mouse button down, which brings up a local menu. Select **Add|Text** from the menu and release. The dialog will appear where you can set the font and text string. Then, left-click on the chart where you want the text to appear.

The text object can be moved into position by left-click-dragging anywhere on the text.

The FinText object does not change position, even if the graph scaling changes.

You can add a trend line programmatically using the FinChartView AddFinTextToPrimaryChart method. **You MUST add it after the FinChartView.BuildChart call.**

Prototypes

C#

```
public FinText AddFinTextToPrimaryChart(
        bool update,
        ChartEvent ev1,
        string text
)
public FinText AddFinTextToPrimaryChart(
        bool update,
        ChartEvent ev1,
        string text,
       int postype
)

public FinText AddFinTextToPrimaryChart(
        bool update,
        string text
)
```

VB

```
 public FinText AddFinTextToPrimaryChart(
        update As Boolean, _
        ev1 As ChartEvent, _
        text As String _
) As FinText

 public FinText AddFinTextToPrimaryChart(
        update As Boolean, _
        ev1 As ChartEvent, _
        text As String _
       postype As Inteter
) As FinText

Public Function AddFinTextToPrimaryChart ( _
        update As Boolean, _
        text As String _
) As FinText
```

Parameters

update
Type: Boolean
Force an immediate update of the chart, rebuilding in the process.

ev1
Type: ChartEvent
The position of the text.

text
Type: String
The text string.

postype
Type: int

Specify whether than text is placed using Normalized Graph Coordinates (ChartObj.NORM_GRAPH_POS) or normalized Plp, ChartObj.NORM_PLOT_PAS).

Return Value

Returns the FinText object created.

Examples

In this example, a FinText object is created in the middle of the current display, and it is up to the user to reposition it using the mouse.

C#

```
this.AddPrimaryChart("TXN",  ChartObj.OHLC);

this.BuildChart();

this.AddFinTextToPrimaryChart(true,"FinText Object");
```

VB

```
Me.AddPrimaryChart("TXN",  ChartObj.OHLC)

Me.BuildChart()

Me.AddFinVLineToPrimaryChart(True ,"FinText Object")
```

In this example, a FinText object is created at a Normalized Plot Area position (0.5, 0.2). The user can still reposition it using the mouse.

C#

```
this.AddPrimaryChart("TXN",  ChartObj.OHLC);

this.BuildChart();

FinText fintext1 = this.AddFinTextToPrimaryChart(true, new Point2D(0.5, 0.2), "Scroll to
the beginning", ChartObj.NORM_PLOT_POS);
```

VB

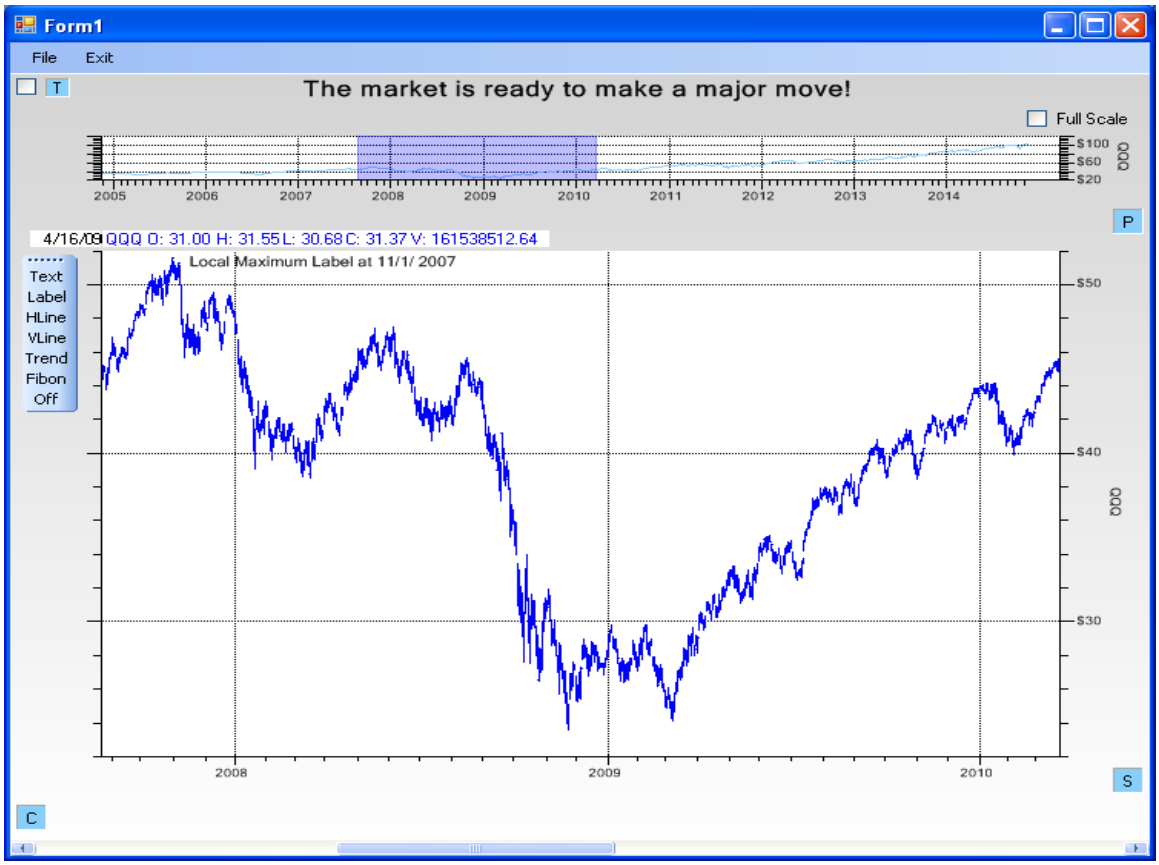
```
Me.BuildChart()

Dim fintext1 As FinText = Me.AddFinTextToPrimaryChart(True, New Point2D(0.5, 0.2),"Scroll
to the beginning", ChartObj.NORM_PLOT_POS)
```

FinLabel

The second type of annotation uses the FinLabel class. It is placed in the chart using Physical graph coordinates. Because of this, it scrolls, and change position, when the graph y-scale is changed, or when the graph is panned or zoomed. Because the FinLabel object sticks to the physical coordinates where it was placed.

The FinLabel object is used to attach annotations to the minimum and maximum values in 2010.

Create a Label annotation for the chart, much the same as a Text annotation. Select the Label option from the tools strip on the right, or from the right-click pop-up menu.



The label object can be moved into position by click-dragging anywhere on the text.

The FinLabel object uses the same dialog box described under the previous FinText description.

You can add a trend line programmatically using the FinChartView AddFinLabelToPrimaryChart method. **You MUST add it after the FinChartView.BuildChart call.**

C#

```
public FinLabel AddFinLabelToPrimaryChart(
        bool update,
        ChartEvent ev1,
        string text
)

public FinLabel AddFinLabelToPrimaryChart(
        bool update,
        string text
)
```

VB

```
Public Function AddFinLabelToPrimaryChart ( _
        update As Boolean, _
        ev1 As ChartEvent, _
        text As String _
) As FinLabel

Public Function AddFinLabelToPrimaryChart ( _
        update As Boolean, _
        text As String _
) As FinLabel
```

Parameters

update
Type: Boolean
Force an immediate update of the chart, rebuilding in the process.

ev1
Type: ChartEvent
The position of the text.

text
Type: String
The text string.

Return Value
Returns the FinLabel object created.

Examples

In this example, a FinLabel object is created in the middle of the current display, and it is up to the user to reposition it using the mouse.

C#

```
this.AddPrimaryChart("TXN",  ChartObj.OHLC);
this.ChartLayoutMode = FinChartView.STANDARD_LAYOUT;

this.BuildChart();

FinLabel finlabel = this.AddFinLabelToPrimaryChart(true,"FinLabel Item");
```

VB

```
Me.AddPrimaryChart("TXN",  ChartObj.OHLC)


Me.BuildChart()

FinLabel finlabel = this.AddFinLabelToPrimaryChart(True,"FinLabel Item");
```

In this example, a FinLabel object is created and placed at the Physical coordinate position represented by stock event (12). The user can still reposition it using the mouse.

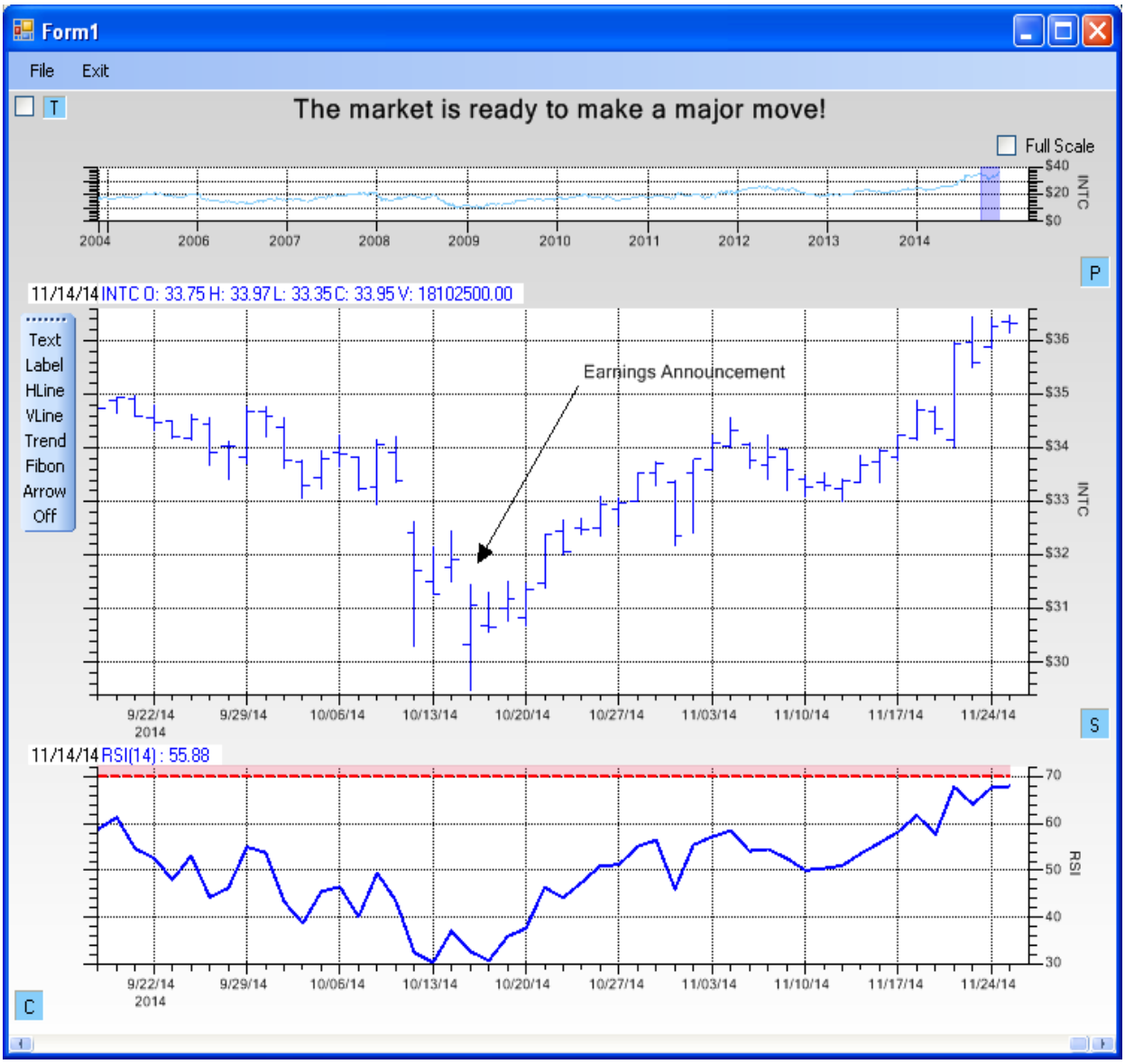
C#

```
this.AddPrimaryChart("TXN",  ChartObj.OHLC);

this.BuildChart();

 FinLabel finlabel = this.AddFinLabelToPrimaryChart(true,
           rawStockData.GetEvent(12),"FinLabel Item");
```

VB

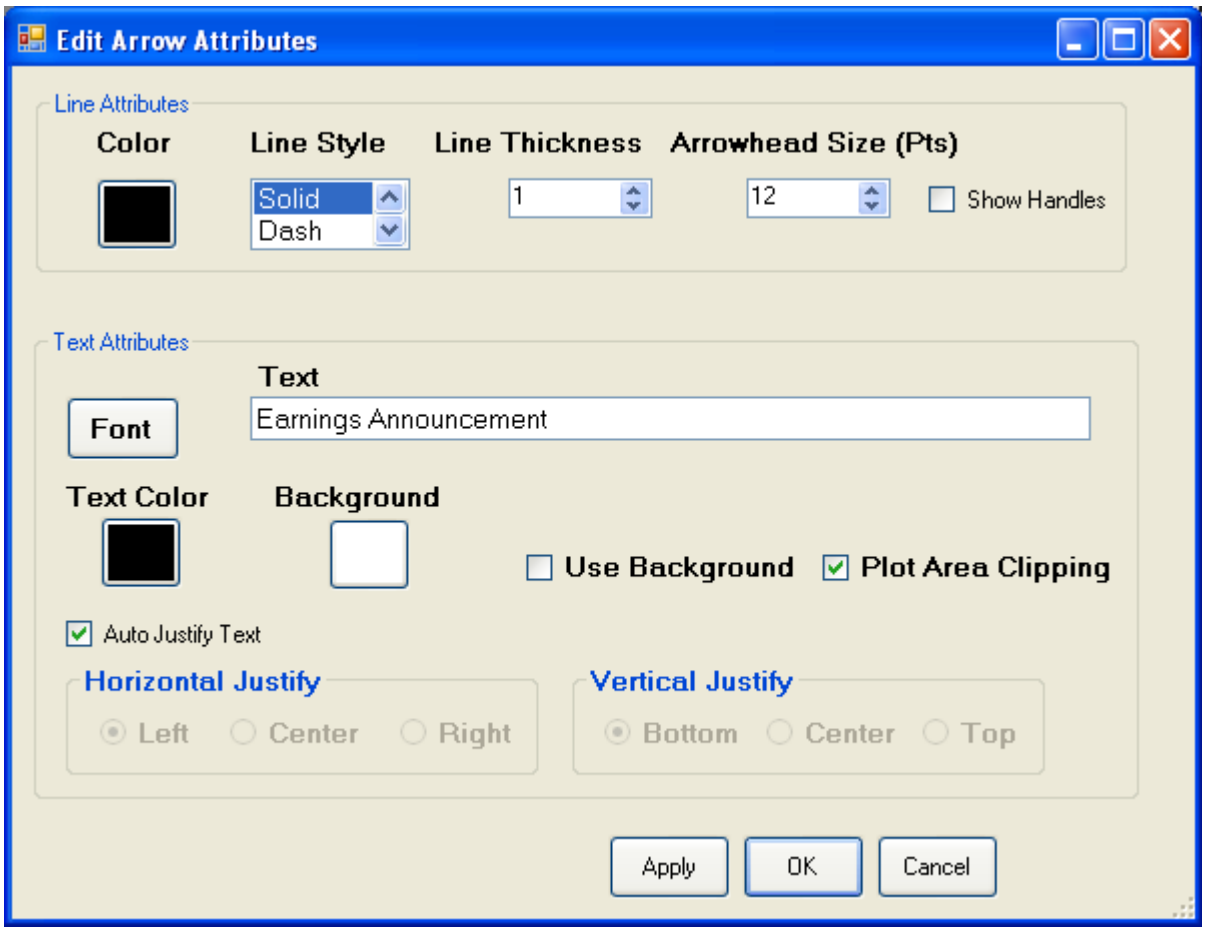
```
Me.AddPrimaryChart("TXN",  ChartObj.OHLC)

Me.BuildChart()

Dim finlabel As FinLabel=
    Me.AddFinLabelToPrimaryChart(True, rawStockData.GetEvent(12),"FinLabel Item")
```

FinArrow



First, select the Arrow option from the toolstrip on the left of the chart. Next, select the arrowhead location by left clicking on the chart at the xy-position where you want the arrow head to appear. Left click again for the tail end of the arrow. The following dialog will appear.

.



Line Attributes

Color – Primary color for the object

Line Style – The line style of the line (Solid, Dash, Dot, DashDot, DashDotDot)

Line Thickness – The line width in pixels of the line – use value in range of (1,2,3,4...15)

A**rrowhead Size** – The size of the arrowhead in points

Show Handles – Check and the grab handles of the trend line are displayed at the endpoints.

Text Attributes

Font – Edit the text Font and Font Size

Text – The text string to display as a label

Text Color – Select the color of the text

Background – Select the background color of the text

Use Background – Check this box if you want the background rectangle of the text to overwrite what is underneath.

Plot Area Clipping – Check this box if you want the text to clip to the plotting area (the area bounded by the axes)

Auto Justify Text – Sets horizontal and vertical text justification of the numeric label depending on the angle of rotation of the arrow

Horizontal Justify – Sets horizontal text justification of the numeric label.

Vertical Justify – Sets vertical text justification of the numeric label.

Set the font you want to use, and the text string and press OK. The text string will appear at the selected xy-position.

You can also define a FinArrow object by right-clicking the mouse on the chart. When you right-click, hold the mouse button down, which brings up a local menu. Select **Add|Arrow** from the menu and release. The dialog will appear where you can set the font and text string. Then, left-click for where the arrowhead should be, and left-click again for where the tail should be.

The FinArrow object can be moved into position by left-click-dragging on one of the endpoints.

You can add a trend line programmatically using the FinChartView AddFinTextToPrimaryChart method. **You MUST add it after the FinChartView.BuildChart call.**

C#

```
public FinArrow AddFinArrowToPrimaryChart(
        bool update,
        ChartEvent ev1,
         ChartEvent ev2,
        string text
)

public FinArrow AddFinArrowToPrimaryChart(
        bool update,
        string text
)
```

VB

```
 public FinArrow AddFinArrowToPrimaryChart(
        update As Boolean, _
        ev1 As ChartEvent, _
         ev2 As ChartEvent, _
        text As String _
) As  FinArrow

Public Function AddFinArrowToPrimaryChart ( _
        update As Boolean, _
        text As String _
) As  FinArrow
```

Parameters

update
Type: Boolean
Force an immediate update of the chart, rebuilding in the process.

ev1
Type: ChartEvent
The arrowhead position in the chart

ev2

Type: ChartEvent
The arrow tail position of the arrow.

text

Type: String
The text string.

Return Value

Returns the FinArrow object created.

Examples

In this case, a default arrow created in the middle of the current display, and it is up to the user to reposition the endpoints of the using the mouse.

C#

```
this.AddPrimaryChart("TXN",  ChartObj.OHLC);


this.BuildChart();

this.AddFinArrowToPrimaryChart(true);
```

VB

```
Me.AddPrimaryChart("TXN",  ChartObj.OHLC)

Me.BuildChart();

Me.AddFinArrowToPrimaryChart(True)
```

In this case, the arrow is defined using OHLC events. The user can still reposition the arrow once it is displayed on the chart.

C#

```
EventGroupDataset rawStockData = this.GetCurrentTickerData();

ChartEvent aev1 = rawStockData.GetEvent(10);
ChartEvent aev2 = rawStockData.GetEvent(15);
FinArrow finarrow1 = this.AddFinArrowToPrimaryChart(true, aev1, aev2, "Look here");
```

VB

```
Dim rawStockData As EventGroupDataset= Me.GetCurrentTickerData()

Dim aev1 As ChartEvent = rawStockData.GetEvent(10)
Dim aev2 As ChartEvent = rawStockData.GetEvent(15)
Dim finarrow1 As FinArrow = Me.AddFinArrowToPrimaryChart(True, aev1, aev2,"Look here")
```

9. Point and Figure Charts

Point and Figure plots have been used in technical analysis for more than 100 years. A P&F chart is unusual in that it does not plot price against time as other techniques do. Instead it plots price against changes in direction. It does so by plotting a column of Xs as the price rises and a column of Os as the price falls. As long the stock price is increasing, and does not backtrack by more than a multiple (usually 3) of the box size, the price increase is displayed as an increasing vertical column of Xs, one X for each time the stock price breaks through the top of a box price level. Once the trend reverses more than a multiple of the box value, the column increments to the right, and changes over to a column of 0's, which are plotted down as long as the stock price continues to drop, without any significant reversals. Electrical and control systems engineers will recognize this as a programmable form of hysteresis. Many technicians like it because it filters out much of the normal up and down noise in the stock data, and makes it very easy to identify trends up or down. As in the example above, it compresses the time frame, so that many years of data (seven in the example above) can be displayed without crowding.

A good book on the subject is *Point and Figure Charting* by Thomas J. Dorsey.

Our version of the Point and Figure charts includes most of the features found in advanced versions. It marks the years of the columns along the x-axis. Months are indicated in-column using numbers 1 to 9 and letters A, B, and C as most implementations do.

Point and Figure y-Axis Scale Modes

There are four common methods used to scale the y-axis: *traditional*, *percentage, fixed,* and *fixed ATR*. The *traditional* mode is the most common. The *traditional* y-axis scale described above,with changes in box size at values of 1, 5, 20, 100..., is generally meant for a stock price range of 5 to 100. It uses mostly integer based numeric values because these are easier to calculate and plot by hand – a legacy of the 100 year history of point and figure charts. But many stocks have a much broader range, especially in non-US markets where the valuation of the stocks in the local currency can be in the thousands. In these cases, a *percentage* based method is preferrable. When the percentage method is used, each box size is a fixed percentage larger than the preceding box size. The two fixed modes (*fixed,* and *fixed ATR)* are useful if the stocks are moving in a relatively narrow range, overlapping one of the traditional mode break-points (1, 5, 20, 100, 200, etc.), and you want to force a constant fixed box size to keep the box size from changing at the breakpoints.

Traditional Box Size

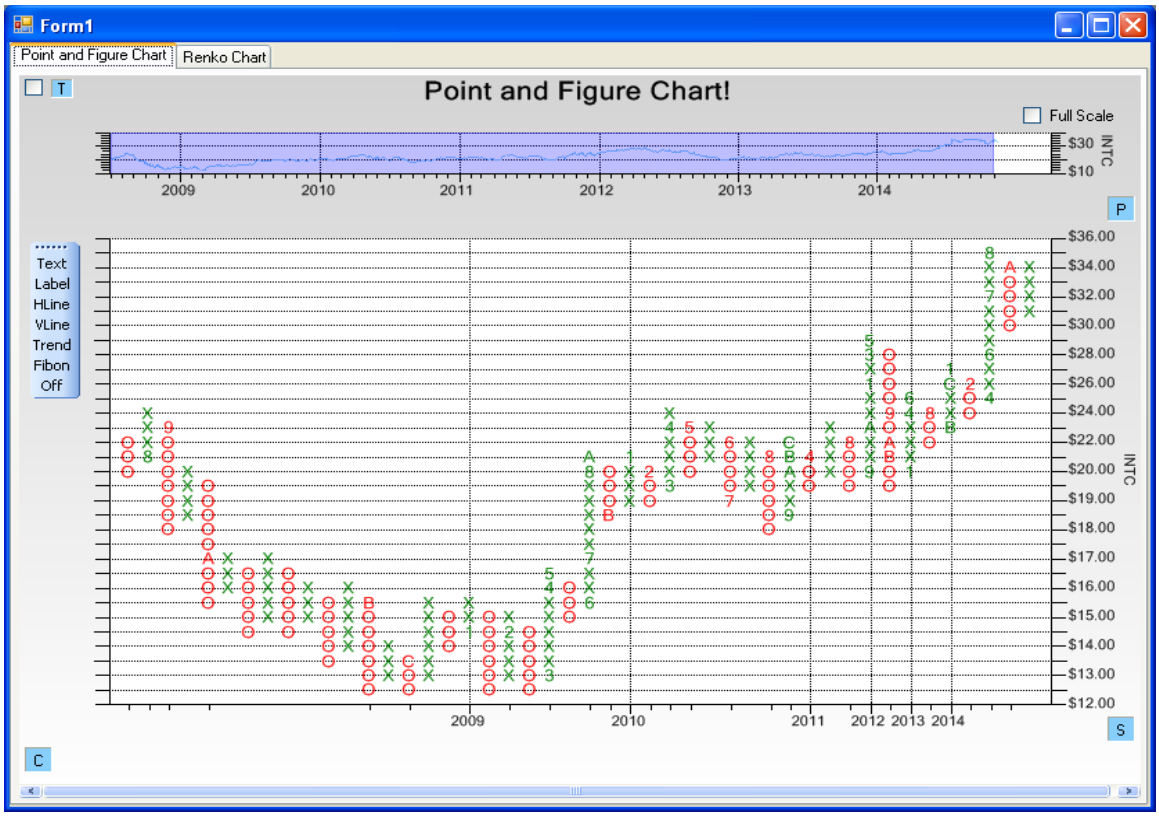
The box-size for the y-axis is increases as the price increase, even within the same column, and follows the traditional box size convention:

Price	**Box size**
0.0 to <=0.25	0.0625
>0.25 to <=1	0.125
>1 to <=5	0.25
>5 to <=20	0.5

>20 to <=100	1.0
>100 to <=200	2.0
>200 to <=500	4.0
>500 to <=1000	5.0
>1000 to <=25000	50.0
> 25000	500.0

In the traditional scaling mode, we use these breakpoints, even though they do not look symetrical across decade ranges.

Under **Price**, the range is exclusive of the first value and inclusive of the second value. So the second range is actually 0.26 to 1.0, and the third range is actually 1.01 to 5.0, and so on.



If you look at the y-axis tick marks in the chart above, you will notice that the in the range of 12 to 20, the box size, represented by the tick spacing, is 0.5. Once the values reach a range greater than 20, the box size shifts to 1.0. This is in accordance with the Price / Box Size table above. If the prices in the chart increased above 100, the box size would change to 2.0.

Percentage Box Size

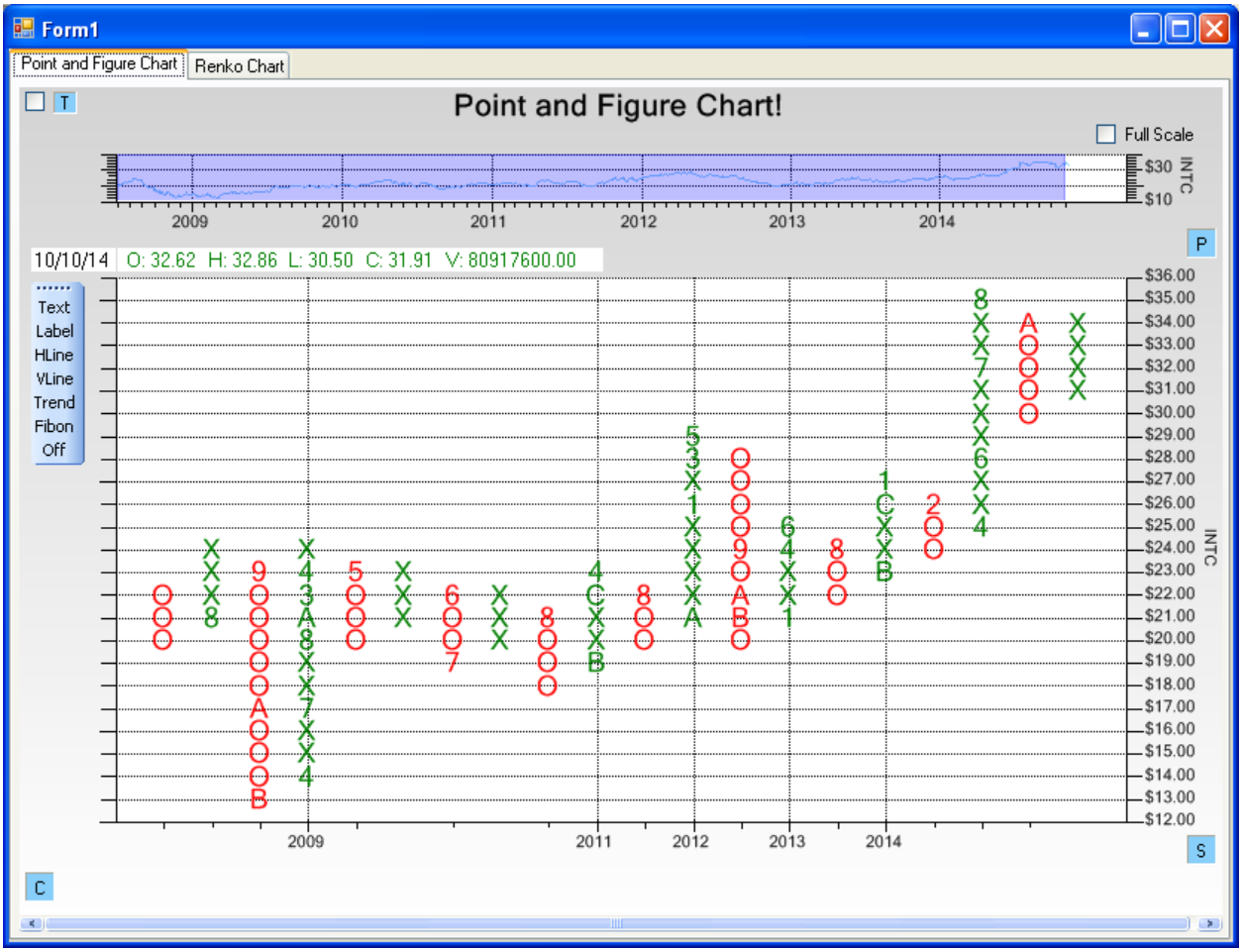
In the percentage method is used, each box size is a fixed percentage larger than the preceding box size. In the chart

below, the P&F chart above has been recast using a percentage y-scale.

Note that the box size, and hence the tick mark spacing, increases by a fixed percentage (3% in this case), from the starting, or minimum y-axis value. This permits charts with a wide dynamic range (Apple for example) to be plotted, keeping the same relative resolution at the minimum and maximum values of the chart.

Fixed Box Size

Another method is to just force the chart to your own choice of a *fixed* box size. In the example below the chart is recast using a fixed box size of 1. The box size can be a whole number (1,2,3,4...) or a fraction (0.5, 0.75, 1.33...)



Fixed Box Size using ATR (Average True Range)

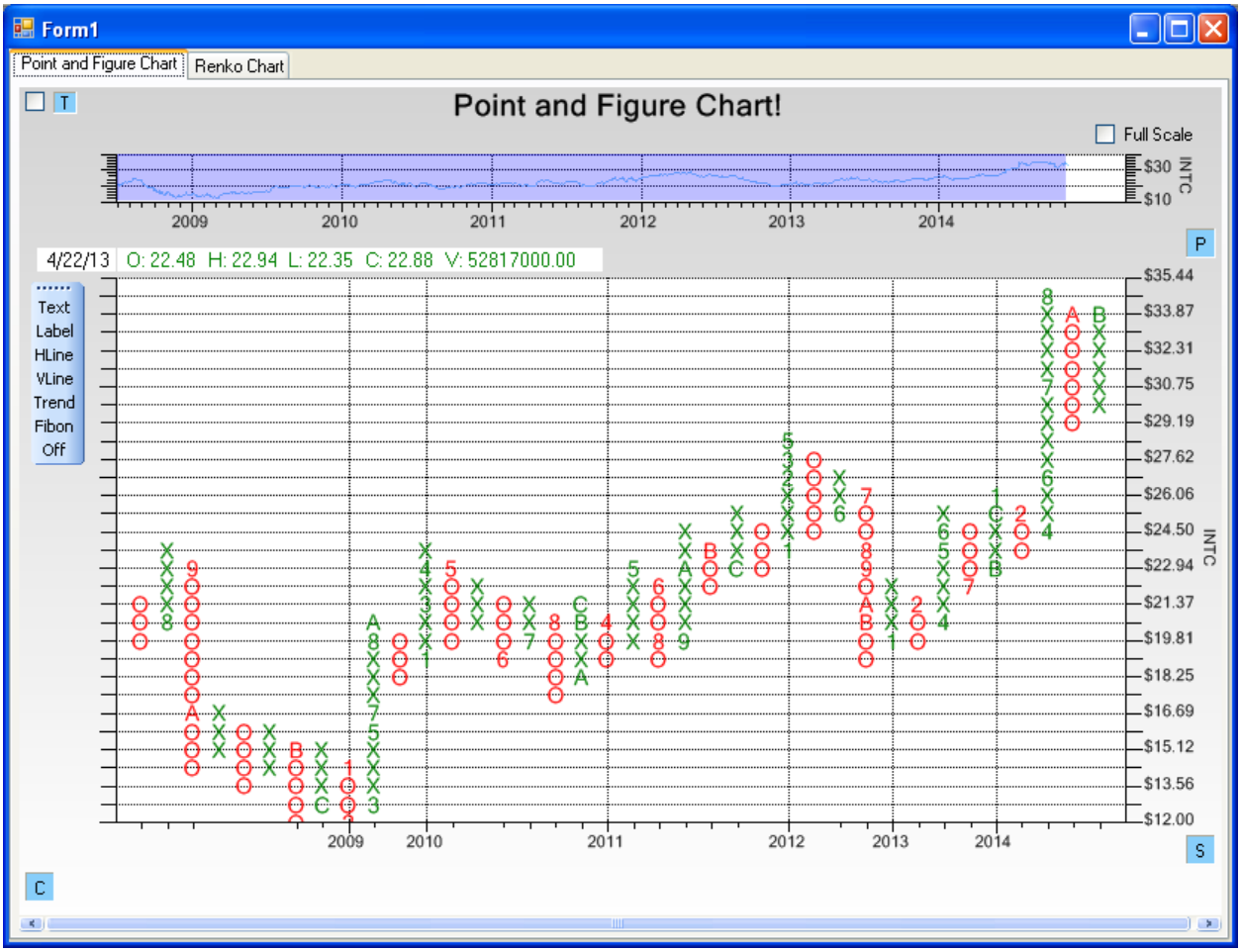
The fixed box size has the drawback in that it is not auto-ranging. A stock where the values in the time frame of interest are in the hundreds (IBM or Apple) will need a much larger fixed box size, than other stocks with values in the teens. The *traditional* and *percentage* methods take this into account with their non-linear scales. A variant of the fixed box size mode calculates fixed box size as a function of the Average True Range, where the Average True Range (defined by J. Welles Wilder) is defined as below.

The true range for a given time period is maximum of several calculated ranges:

True Range = Max (Current High less the current Low,

Abs(Current High - previous Close),

Abs(Current Low - previous Close))

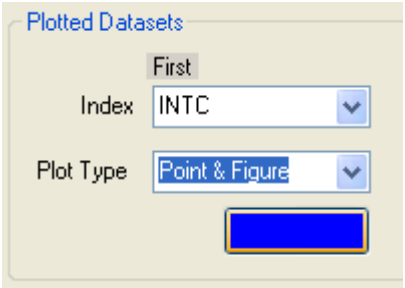


The *Average True Range* is a moving average (14-day Moving Average in this case) of the most recent and pervious 13 values. The fixed box size is set to the value of the Average True Range calculated using the most recent data. Below is the P&F chart we have been using as an example, configured for a fixed box size calculated using the Average True Range.

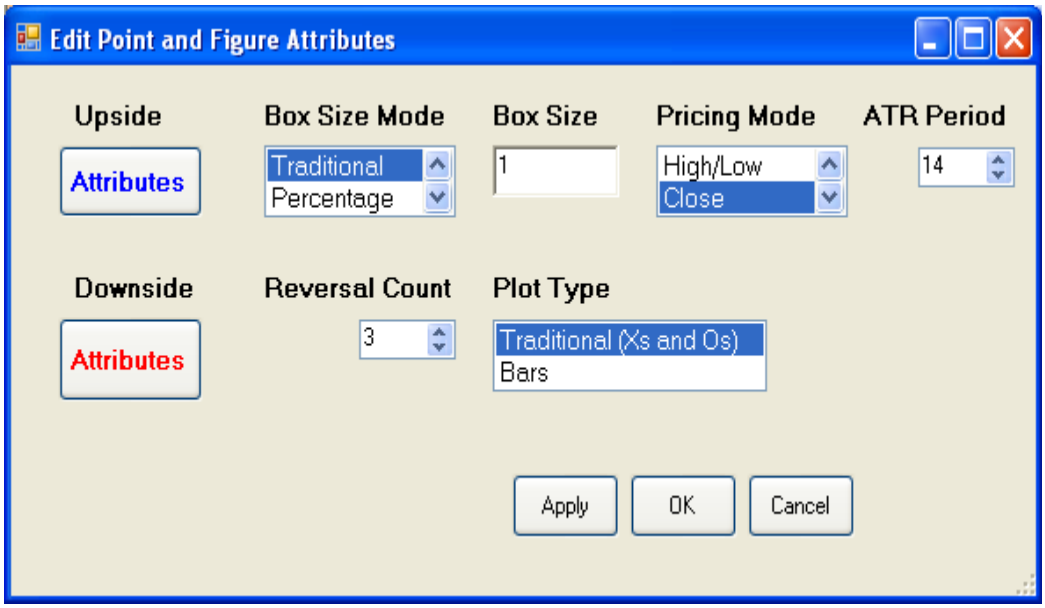
In this case the fixed box size was calculated to be approximately 0.78.

Point And Figure Dialog Box

The P&F box size mode can be set using the Attribute dialog for the chart, found in the Primary Chart Dialog page.



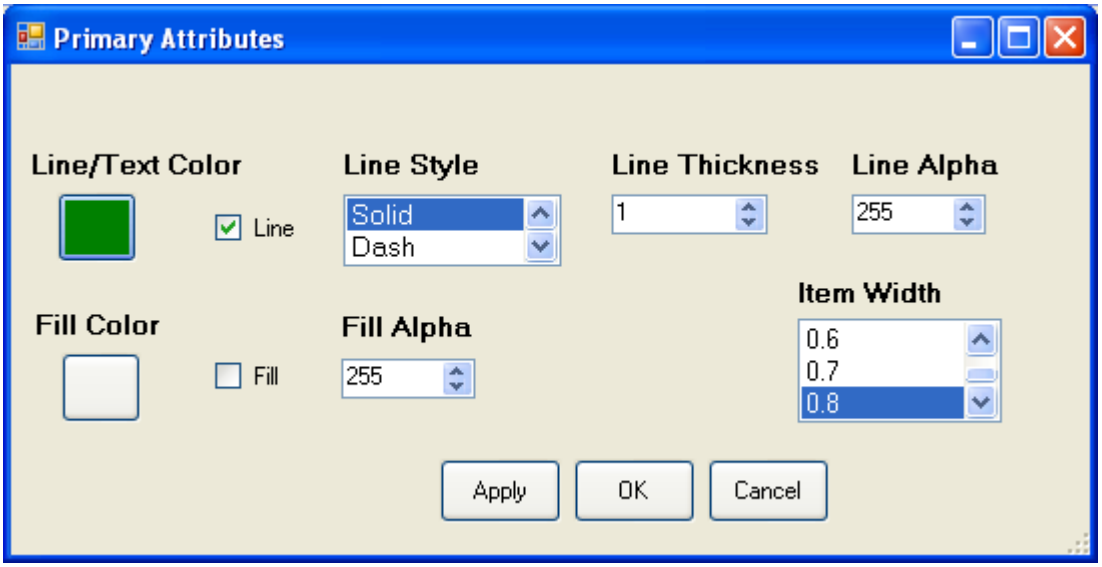
Select the Attribute box (the blue box above) and the P&F dialog will appear.



Point And Figure Colors

Select the Upside and/or Downside Attributes buttons to change the Point and Figure colors.

At that point an Attributes Dialog box will appear.



When working with the Traditional Xs and Os Point and Figure plot, you want the text **Line/Text Color** set to the color you want for the X's and Os. The Fill mode should be left unchecked. The other fields should remain unchanged.

In code, this looks like:

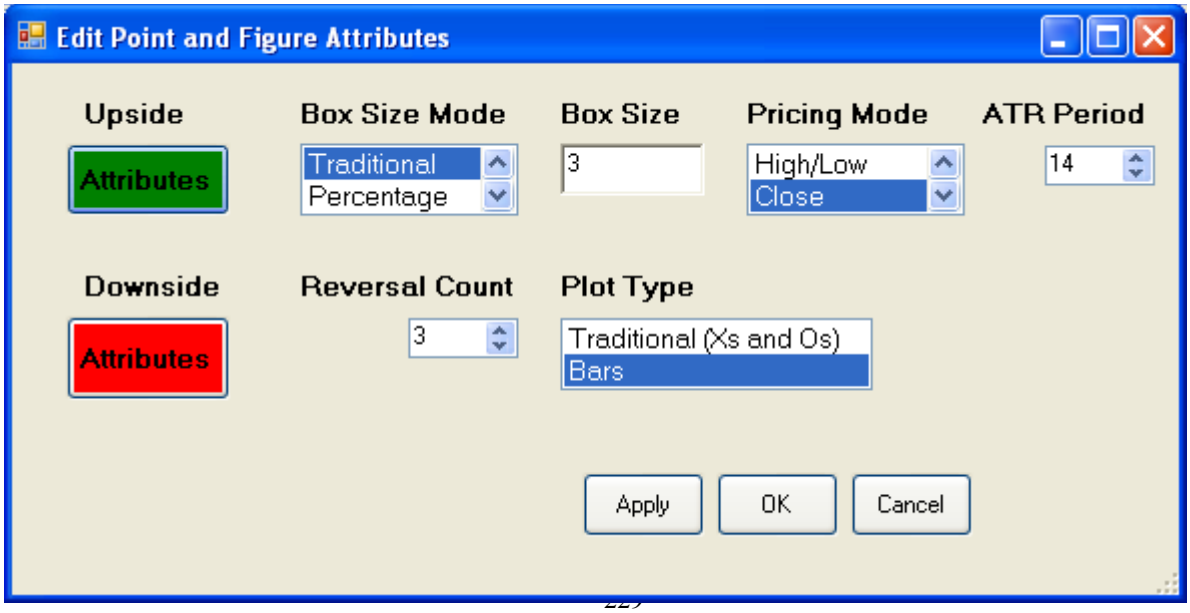
C#

```
FinPointAndFigureChartPlot plotobj = this.AddPrimaryChartPointAndFigure();

plotobj.PlotUpAttribute = new ChartAttribute(Color.Green, 1, DashStyle.Solid);
plotobj.PlotDownAttribute = new ChartAttribute(Color.IndianRed, 1, DashStyle.Solid);
```

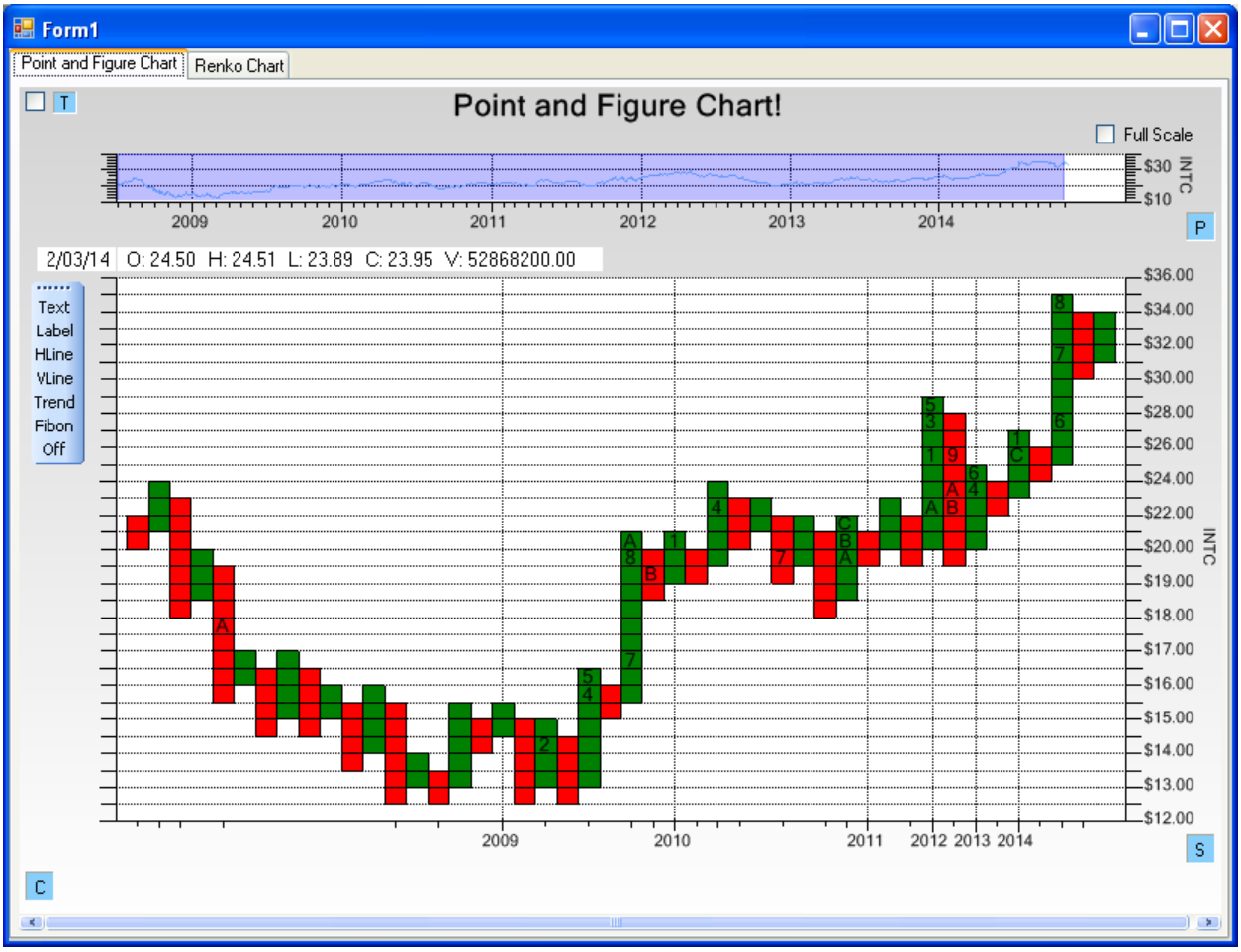
VB

```
Dim plotobj As FinPointAndFigureChartPlot= Me.AddPrimaryChartPointAndFigure()

plotobj.PlotUpAttribute = new ChartAttribute(Color.Green, 1, DashStyle.Solid)
plotobj.PlotDownAttribute = new ChartAttribute(Color.IndianRed, 1, DashStyle.Solid)
```

If you plan to use the bar version of the Point and Figure plot, you will need to set the Line/Text Color to Black, and the Fill Color to the color your want; also check the Fill box.



The resulting chart will look like:



In code, this looks like:

C#

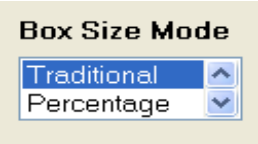
```
FinPointAndFigureChartPlot plotobj = this.AddPrimaryChartPointAndFigure();

plotobj.PlotUpAttribute = new ChartAttribute(Color.Black, 1, DashStyle.Solid,
Color.Green);
plotobj.PlotDownAttribute = new ChartAttribute(Color.Black , 1, DashStyle.Solid,
Color.IndianRed);
```

VB

```
Dim plotobj As FinPointAndFigureChartPlot= Me.AddPrimaryChartPointAndFigure()

plotobj.PlotUpAttribute = new ChartAttribute(Color.Black, 1, DashStyle.Solid, Color.Green)
plotobj.PlotDownAttribute = new ChartAttribute(Color.Black , 1, DashStyle.Solid,
Color.IndianRed)
```

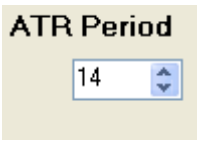
Box Size Mode

The **Box Size Mode** combo box is used to select the P&F box size mode: Traditional, Percentage, Fixed, or Fixed ATR.



If you select Traditional or Fixed ATR, the **Box Size** field is ignored. If you select a Box Size Mode of Percentage, then the percentage used in the calculation is the value of Box Size converted to percent. For example, a Box Size of 1 represents 1 percent (0.01). A value of 0.5 represents 0.5 percent (0.005). If you select a box Size of Fixed, the box size is set to the value of Box size.

If the Box Size Mode is set to Fixed ATR, the ATR look-back period is specified using the **ATR Period** combo box, seen below.



If you want to set the **Box Size Mode** property under program control, set the appropriate FinPointAndFigureChartPlot property, as returned by the AddPrimaryChartPointAndFigurecall in your program.

C#

```
FinPointAndFigureChartPlot plotobj = this.AddPrimaryChartPointAndFigure();

plotobj.BoxSizeMode = FinChartConstants.POINT_AND_FIGURE_YSCALE_PERCENTAGE;
```

VB

```
Dim FinPointAndFigureChartPlot As plotobj = Me.AddPrimaryChartPointAndFigure()

plotobj.BoxSizeMode = FinChartConstants.POINT_AND_FIGURE_YSCALE_PERCENTAGE
```

Specify the **BoxSizeMode** using one of the following FinChartConstants constants:

```
FinChartConstants.POINT_AND_FIGURE_YSCALE_TRADITIONAL
FinChartConstants.POINT_AND_FIGURE_YSCALE_PERCENTAGE
FinChartConstants.POINT_AND_FIGURE_YSCALE_FIXED
FinChartConstants.POINT_AND_FIGURE_YSCALE_ATRFIXED
```

If you choose the POINT_AND_FIGURE_YSCALE_FIXED box size mode, you can set the related fixed box property using the BoxSize property.

```
plotobj.BoxSize = 3;
```

If you choose the POINT_AND_FIGURE_YSCALE_ATRFIXED box size mode, you can set the related ATR look-back period using the ATRPeriod property.

```
plotobj.ATRPeriod = 14
```

Point And Figure Price Mode

The price input to the chart takes one of three forms. You can plot the chart using Close prices only. Or you can use the most popular method which uses the days High and Low values. The third method is similar to the first option, but instead of using the Close price, the Typical Price is used, where the Typical price is defined as: (High value + Low value + Close value) / 3.

High Low Method

Here is the basic psuedo-code algorithm for the High-Low method.

```
IF (you are currently in an X-column, i.e. prices have been rising)
{
        IF  (High breaks out of the current box into a new, higher box)
                draw the new X and ignore the Low
        ELSE    IF (Low triggers a 3-box reversal)
                Start a new column and fill in Os down to to current Low
        ELSE
                Do nothing
}
ELSE IF (you are currently in an O-column, i.e. prices have been falling)
{
        IF  (Low breaks out of the current box into a new, lower box)
                draw the new O 's
        ELSE    IF (High triggers a 3-box reversal)
                Start a new column and fill in Xs up  to to current High
        ELSE
                Do nothing
}
```

Close Method

Here is the basic psuedo-code algorithm for the Close method

```
IF (you are currently in an X-column, i.e. prices have been rising)
{
        IF  (Close breaks out of the current box into a new, higher box)
```

```
                draw the new X
        ELSE    IF ( Close triggers a 3-box reversal)
                Start a new column and fill in Os down to to current Close
        ELSE
                Do nothing
}
ELSE IF (you are currently in an O-column, i.e. prices have been falling)
{
        IF  ( Close breaks out of the current box into a new, lower box)
                draw the new O 's
        ELSE    IF ( Close triggers a 3-box reversal)
                Start a new column and fill in Xs up  to to current Close
        ELSE
                Do nothing
}
```

Typical Price Method

The Typical Price for each OHLC event is calculated as: (High value + Low value + Close value) / 3.

Here is the basic psuedo-code algorithm for the Typical Price method

```
IF (you are currently in an X-column, i.e. prices have been rising)
{
        IF  ( Typical Price breaks out of the current box into a new, higher box)
                draw the new X
        ELSE    IF (  Typical Price triggers a 3-box reversal)
                Start a new column and fill in Os down to to current  Typical Price
        ELSE
                Do nothing
}
ELSE IF (you are currently in an O-column, i.e. prices have been falling)
```

{

IF (Typical Price breaks out of the current box into a new, lower box)

draw the new O 's

ELSE IF (Typical Price triggers a 3-box reversal)

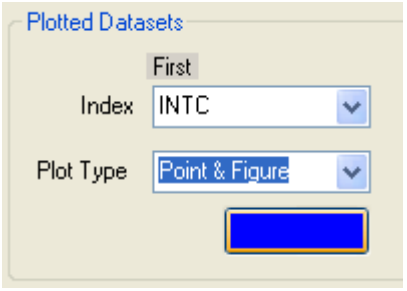
Start a new column and fill in Xs up to to current Typical Price

ELSE

Do nothing

}

The P&F Price mode can be set using the Attribute dialog for the chart, found in the Primary Chart Dialog page.



Select the Attribute box (the blue box above) and the P&F dialog will appear. Set the Pricing Mode combon box to the value you want.

If you want to set the **Pricing Mode** property under program control, set the appropriate FinPointAndFigureChartPlot parameter, as returned by the AddPrimaryChartPointAndFigurecall in your program.

C#

```
FinPointAndFigureChartPlot plotobj = this.AddPrimaryChartPointAndFigure();

plotobj.PricingMode = FinChartConstants.POINT_AND_FIGURE_PRICE_CLOSE;
```

VB

```
Dim plotobj As FinPointAndFigureChartPlot = Me.AddPrimaryChartPointAndFigure()
```

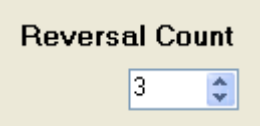
```
plotobj.PricingMode = FinChartConstants.POINT_AND_FIGURE_PRICE_CLOSE
```

Specify the **PricingMode** using one of the following FinChartConstants constants:

```
FinChartConstants.POINT_AND_FIGURE_PRICE_CLOSE
FinChartConstants.POINT_AND_FIGURE_PRICE_HIGHLOW
FinChartConstants.POINT_AND_FIGURE_PRICE_TYPICAL
```

Reversal Count

The *reversal count* is the number of boxes the price of a stock must change, in order to trigger a P&F reversal. A reversal is when the chart transitions from Xs and Os, (and vise-versa), and a new column is started in the chart. Typically this value is three, but it can be anything from 1 to N.



The reversal count is set using the **Reversal Count** field of the Point and Figure Attributes dialog box.

If you want to set the **Reversal Count** property under program control, set the appropriate FinPointAndFigureChartPlot parameter, as returned by the AddPrimaryChartPointAndFigurecall in your program.

C#

```
FinPointAndFigureChartPlot plotobj = this.AddPrimaryChartPointAndFigure();
```

VB

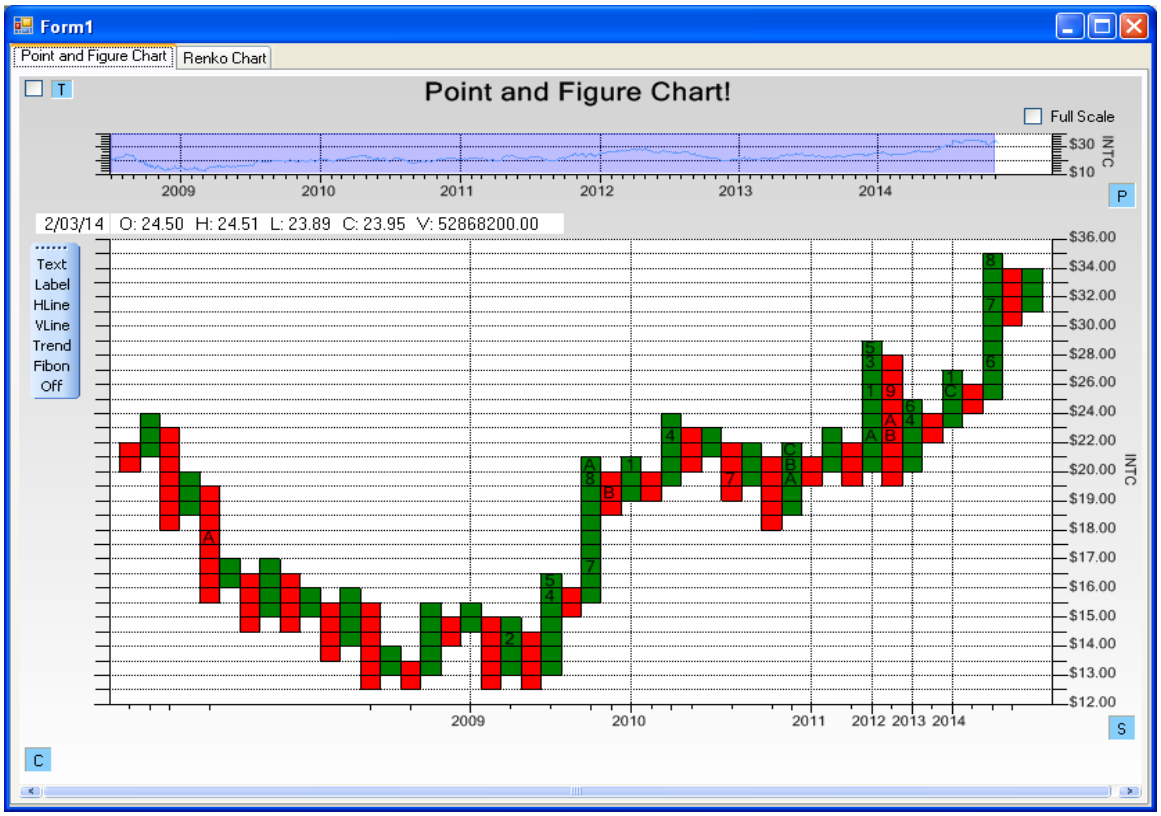
```
Dim plotobj As FinPointAndFigureChartPlot = Me.AddPrimaryChartPointAndFigure()
```

Specify the **ReversalCount** value to value reasonable to your range of data, an integer in the range 1 to 10.

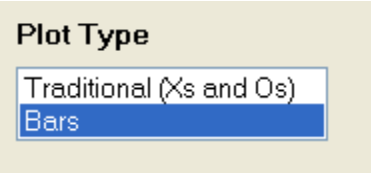
```
plotobj.ReversalCount = 2;
```

Plot Type

We created a Point and Figure variant which uses colored bars, instead of Xs and O's. See the example below.



The plot type (Traditional or Bars) t is set using the Plot Type combo box of the Point and Figure Attributes dialog box.



If you want to set the **Plot Type** property under program control, set the appropriate FinPointAndFigureChartPlot property, as returned by the AddPrimaryChartPointAndFigurecall in your program.

C#

```
FinPointAndFigureChartPlot plotobj = this.AddPrimaryChartPointAndFigure();

plotobj.PfPlotType = FinChartConstants.POINT_AND_FIGURE_PLOT_TRADITIONAL;
```

VB

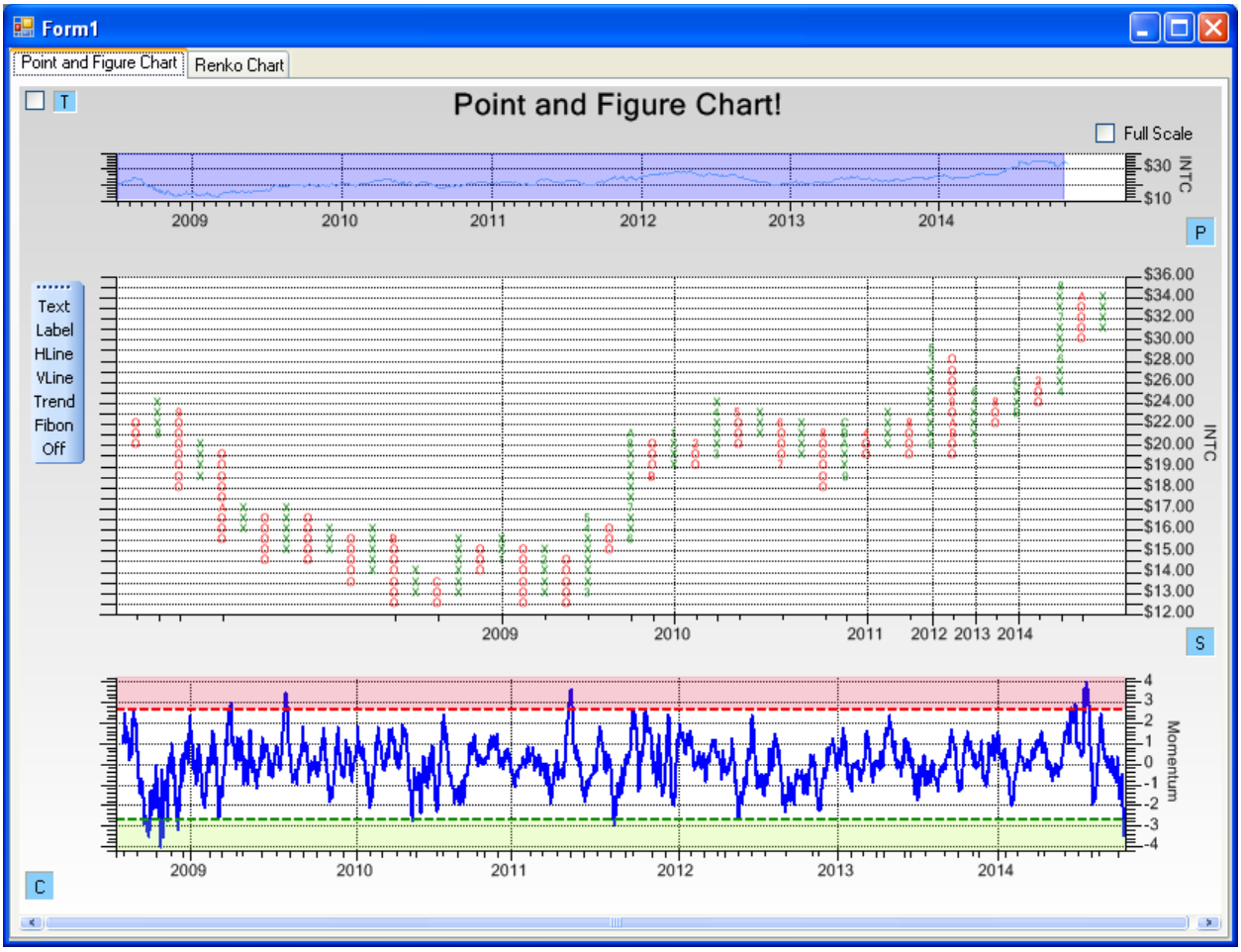
```
FinPointAndFigureChartPlot plotobj = Me.AddPrimaryChartPointAndFigure()

plotobj.PfPlotType = FinChartConstants.POINT_AND_FIGURE_PLOT_TRADITIONAL
```

Specify the **PfPlotType** value to one of the following FinChartConstants constants:

```
FinChartConstants.POINT_AND_FIGURE_PLOT_TRADITIONAL
FinChartConstants.POINT_AND_FIGURE_PLOT_BARS
```

Combining Point and Figure Charts with Secondary Chart Technical Indicators

The Point and Figure plot (and the Renko plot) is a special case in the FinChartView Primary chart. This is because it uses a unique coordinate system which does not match up with the more traditional linear sequential event-based coordinate system used in the standard FinChartView Primary plot types (Line, OHLC, Candlestick, Bar, and Mountain). As you see in the examples, the coordinate system is non-linear with respect to time in the x-axis, and also non-linear in the y-axis (in the Traditional and Percentage modes). Since its x-axis coordinate system is unique to the underlying data, it will not sync, exactly (when zooming, panning and scrolling) with the other Secondary chart technical indicator plots displayed in the FinChartView. Instead, only the starting and ending dates will sync. All other dates between the endpoints will be out of sync.



When a Point and Figure chart, or a Renko chart, is displayed in the Primary chart window, the technical indicators in the secondary window(s) will only be in-sync at the endpoints.

Creating a Point and Figure chart

The Point and Figure charting routines are integrated in the FinChartView class. So, you create a FinChartData object with a portfolio of stocks, use that to initialize the FinChartView, and then add a Point and Figure chart as the Primary Chart for the FinChartView, using AddPrimaryChartPointAndFigure. See the example PointAndFigureExample.PointAndFigureUserControl1.

C#

```
public void InitializeChart()
{
  this.PreferredSize = new Size(800, 600);
  String[] idStrings = { "BA", "INTC", "IBM", "TXN", "AMAT", "CSCO" };
  String[] tickerStrings = { "BA", "INTC", "IBM",
                             "TXN", "AMAT", "CSCO" };

  ChartCalendar startdate = new ChartCalendar(2008, ChartObj.JULY, 2);
  ChartCalendar enddate = new ChartCalendar(2014, ChartObj.AUGUST, 25);

  FinYahooURLCurrentDataSource finStockData = new FinYahooURLCurrentDataSource();
  FinYahooURLHistoricalDataSource finStockHistoricalData =
        new FinYahooURLHistoricalDataSource();

  for (int i = 0; i < idStrings.Length; i++)
  {
    finStockHistoricalData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
    finStockData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
  }
  FinChartData finChartData = new FinChartData(finStockHistoricalData,
     finStockData, idStrings, startdate, enddate);

  // Add all of the column items to the table
  finChartData.AddAllColumnItems();
  // Initially turn the table off.
  this.EnableFinChartTable = false;

  // Assign FinChartView's data
  InitFinChartView(finChartData);

  // only needed if you are reinitalizing the charts
  this.ResetTechnicalCharts();
  // main title
  this.MainTitleString = "Point and Figure Chart!";
   // Choose the stock for the initial chart,
   // from the stocks entered into finChartData
   this.CurrentTickerString = "INTC";
   // Set zoom flag true
   FinPointAndFigureChartPlot plotobj = this.AddPrimaryChartPointAndFigure();

   plotobj.PfPlotType = FinChartConstants.POINT_AND_FIGURE_PLOT_TRADITIONAL;
   // Build the chart using the given parameters
   this.BuildChart();

}
```

VB

```
Public Sub InitializeChart()
 Me.PreferredSize = New Size(800, 600)
 Dim idStrings As [String]() = {"BA","INTC","IBM","TXN","AMAT","CSCO"}
 Dim tickerStrings As [String]() = {"BA","INTC","IBM","TXN","AMAT","CSCO"}

 Dim startdate As New ChartCalendar(2008, ChartObj.JULY, 2)
' ChartCalendar enddate = new ChartCalendar(2014, ChartObj.AUGUST, 25);
 Dim enddate As New ChartCalendar()
' Todday
 Dim finStockData As New FinYahooURLCurrentDataSource()
 Dim finStockHistoricalData As New FinYahooURLHistoricalDataSource()
```

```
 For i As Integer = 0 To idStrings.Length - 1
    finStockHistoricalData.AddTickerLookupItem(idStrings(i), tickerStrings(i))
    finStockData.AddTickerLookupItem(idStrings(i), tickerStrings(i))
 Next
 Dim finChartData As New FinChartData(finStockHistoricalData, finStockData, idStrings,
startdate, enddate)

' Add all of the column items to the table
 finChartData.AddAllColumnItems()
' Initially turn the table off.
 Me.EnableFinChartTable = False

' Assign FinChartView's data
 InitFinChartView(finChartData)

' only needed if you are reinitalizing the charts
 Me.ResetTechnicalCharts()
' main title
 Me.MainTitleString ="Point and Figure Chart!"
' Choose the stock for the initial chart, from the stocks entered into finChartData
 Me.CurrentTickerString ="INTC"

 Dim plotobj As FinPointAndFigureChartPlot = Me.AddPrimaryChartPointAndFigure()
' Assigne the up and down colors
 plotobj.PlotUpAttribute = New ChartAttribute(Color.Green, 1, DashStyle.Solid,
Color.Green)
 plotobj.PlotDownAttribute = New ChartAttribute(Color.Red, 1, DashStyle.Solid, Color.Red)
 plotobj.BoxSize = 3
 plotobj.BoxSizeMode = FinChartConstants.POINT_AND_FIGURE_YSCALE_TRADITIONAL
' plotobj.BoxSizeMode = FinChartConstants.POINT_AND_FIGURE_YSCALE_PERCENTAGE;
' plotobj.BoxSizeMode = FinChartConstants.POINT_AND_FIGURE_YSCALE_FIXED;
' plotobj.BoxSizeMode = FinChartConstants.POINT_AND_FIGURE_YSCALE_ATRFIXED;

'plotobj.PricingMode = FinChartConstants.POINT_AND_FIGURE_PRICE_CLOSE;
 plotobj.PricingMode = FinChartConstants.POINT_AND_FIGURE_PRICE_HIGHLOW
' plotobj.PricingMode = FinChartConstants.POINT_AND_FIGURE_PRICE_TYPICAL;

 plotobj.ReversalCount = 3

 plotobj.ATRPeriod = 14
 plotobj.PfPlotType = FinChartConstants.POINT_AND_FIGURE_PLOT_TRADITIONAL
' plotobj.PfPlotType = FinChartConstants.POINT_AND_FIGURE_PLOT_BARS;
' Build the chart using the given parameters
 Me.BuildChart()

 End Sub
```

If you want to ditch the Xs and Os, and go with the alternate version which uses filled bars instead, use the following code instead. Set the BoxFillAttributes property with the second color for the bars, and the PfDisplayMode property to 1.

A Point and Figure chart using bars, rather than Xs and Os.

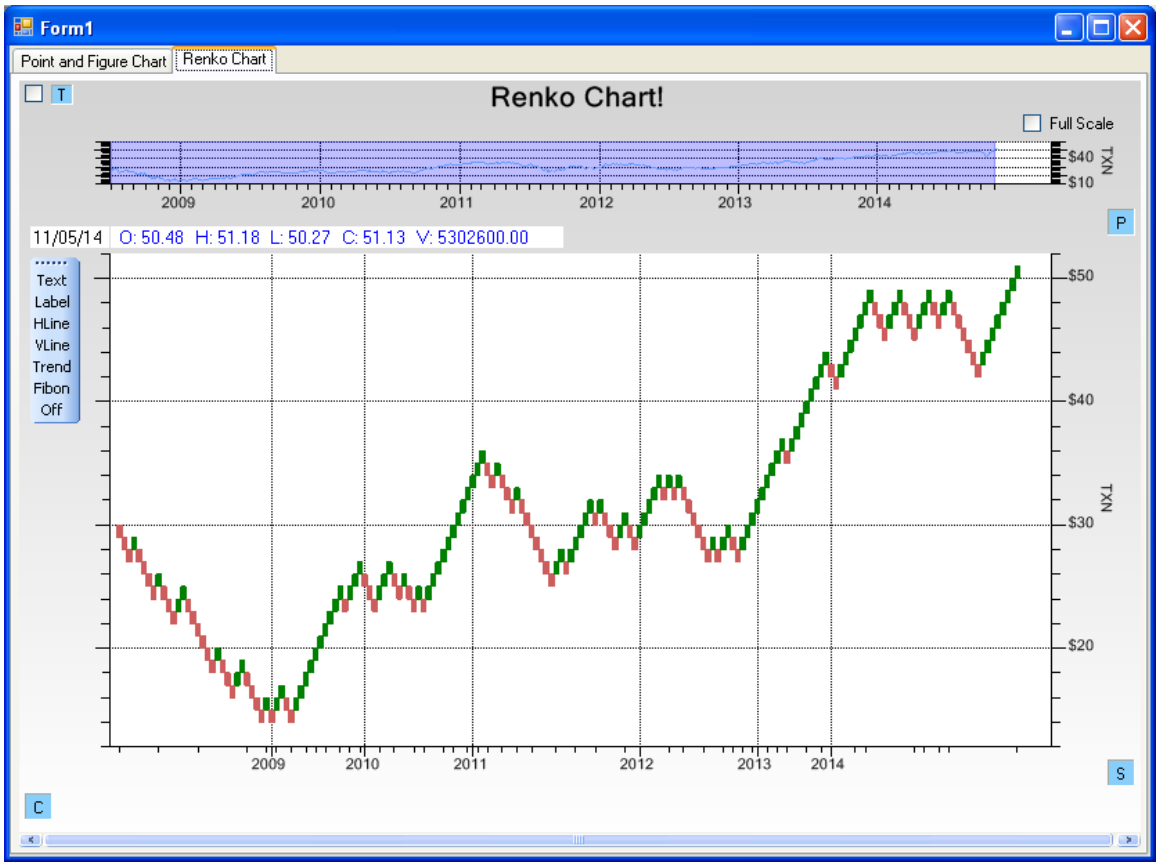
C#

```
plotobj.PfPlotType = FinChartConstants.POINT_AND_FIGURE_PLOT_TRADITIONAL;
// plotobj.PfPlotType = FinChartConstants.POINT_AND_FIGURE_PLOT_BARS;
```

VB

```
plotobj.PfPlotType = FinChartConstants.POINT_AND_FIGURE_PLOT_TRADITIONAL
' plotobj.PfPlotType = FinChartConstants.POINT_AND_FIGURE_PLOT_BARS
```

10. Renko Charts



Renko charts are similar to Point and Figure charts, in that they do not plot price against time as other techniques do. Instead it plots price against changes in direction by plotting filled boxes (called *bricks* in Renko terminology) as the price rises and unfilled boxes as the price falls. As long the stock price is increasing, and does not backtrack by more than the brick size size, the price increase is displayed as an rising diagonal of bricks. Each time the price rises enough to warrant a new brick, a new column is started and the brick is plotted in that column. Once the trend reverses, as each new brick is added on the downside, a new column is started. No column will every contain more than one brick. The result is a chart similar to the example above, where time scales are irregular and compressed. The net result is a strong filtering of the OHLC data, eliminating the ever present noise present in market data.

Special Note

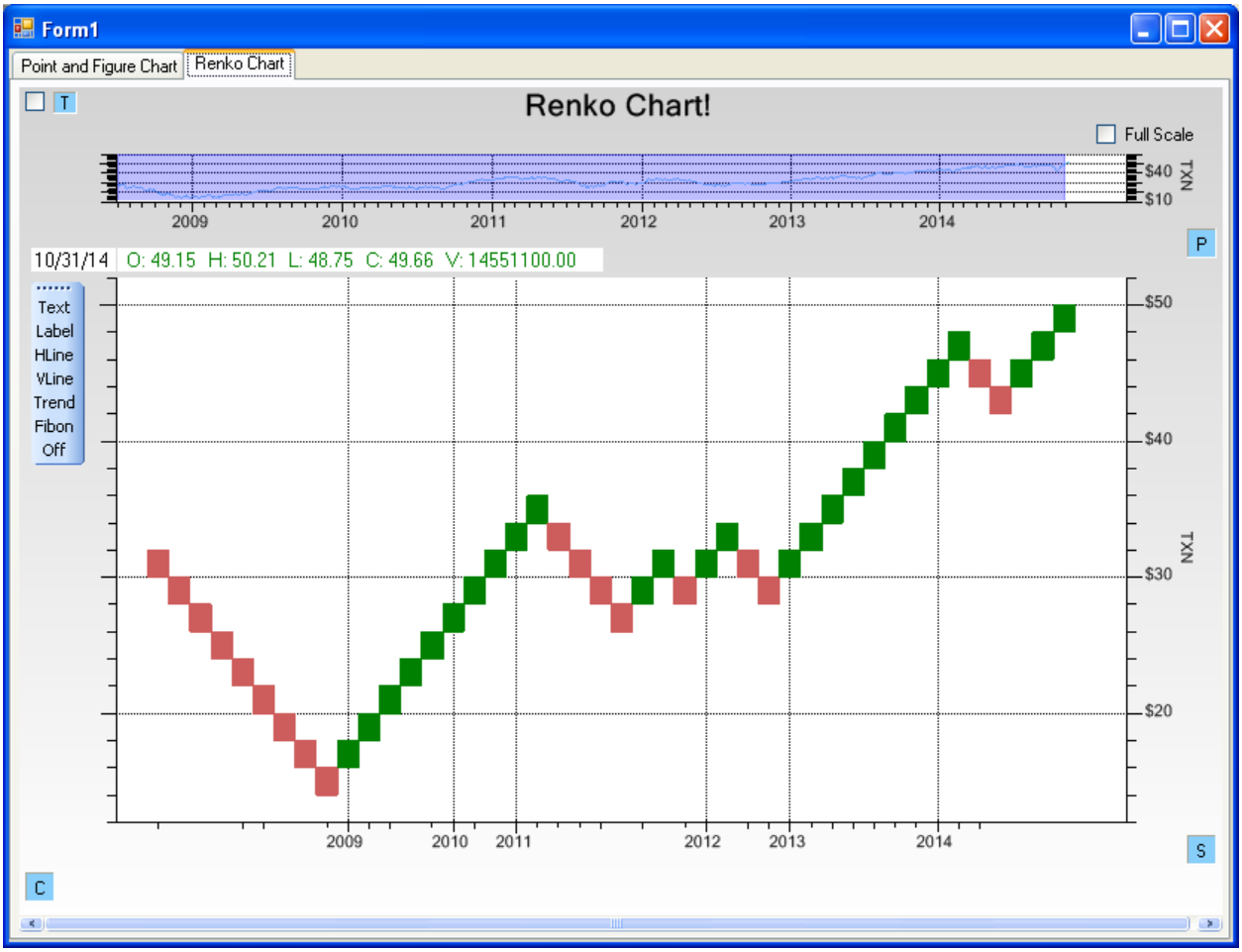
You can display the Renko Chart in the Primary Chart of the FinChartView window. At the same time, you can display the Technical Indicator charts in the Secondary Chart windows. Since the Renko chart uses a different x-axis scale than the Technical Indicator charts, the Primary and Secondary x-axis scales will **not** line up date for date. Only the starting and ending dates will match. Everything else in between will be different.

Renko y-Axis Scale Modes

Unlike Point and Figure charts, which in the Traditional and Percentage scale modes use a non-linear y-axis, Renko charts always use a linear y-axis scale. Also, Renko charts always use a fixed box size. The box size can be set by the user, or it can be automatically calculated using the ATR for a given look-back period.

Fixed Box Size

The user has the option of specifying a fixed box sise. In the example below the chart is recast using a fixed box size of 2. The box size can be a whole number (1,2,3,4...) or a fraction (0.5, 0.75, 1.33...)



Fixed Box Size using ATR (Average True Range)

The fixed box size has the drawback in that it is not auto-ranging. A stock where the values in the time frame of interest are in the hundreds (IBM or Apple) will need a much larger fixed box size, than other stocks with values in the teens. A variant of the fixed box size mode calculates fixed box size as a function of the Average True Range, where the Average True Range (defined by J. Welles Wilder) is defined as below.

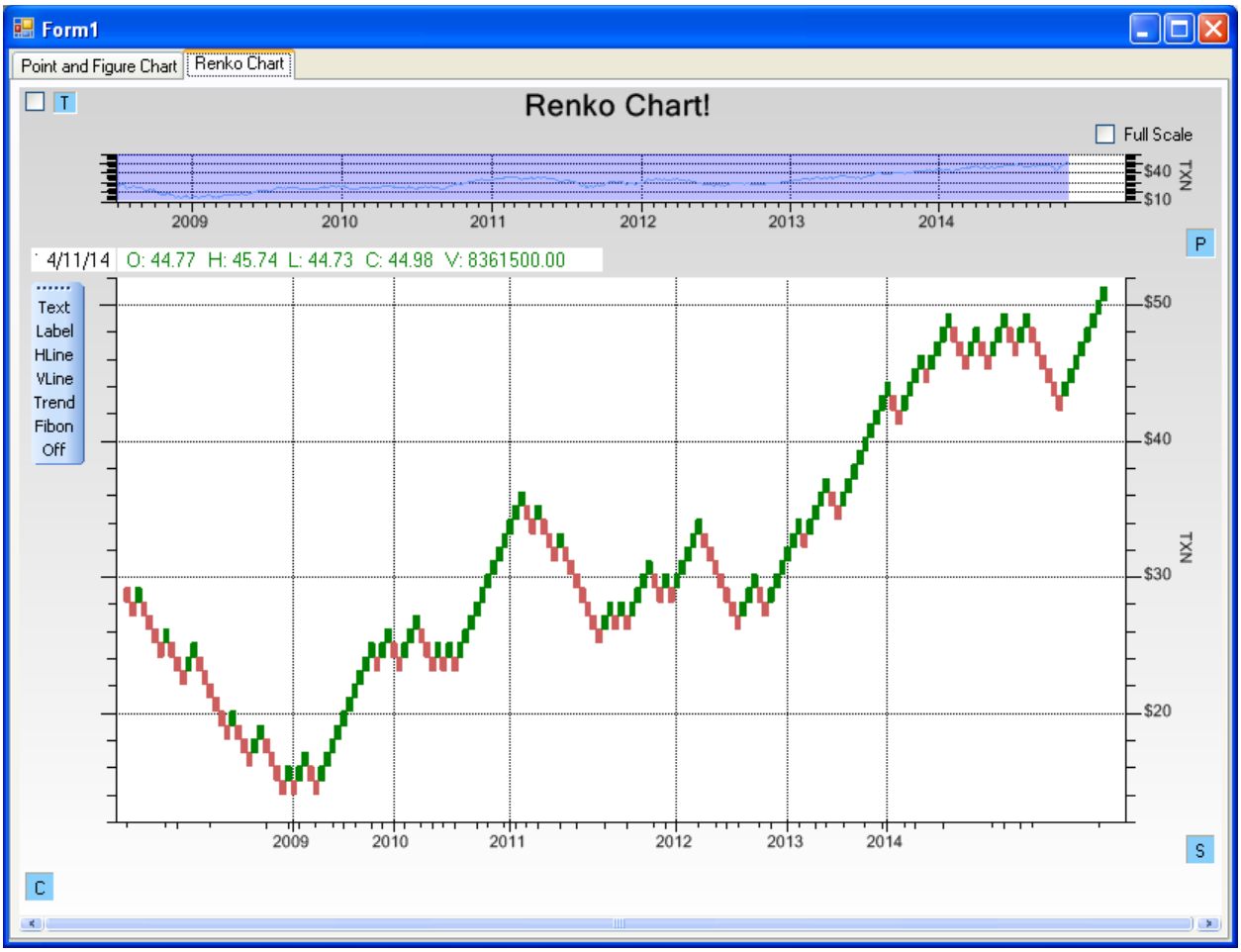
The true range for a given time period is maximum of several calculated ranges:

True Range = Max (Current High less the current Low,

Abs(Current High - previous Close),

Abs(Current Low - previous Close))

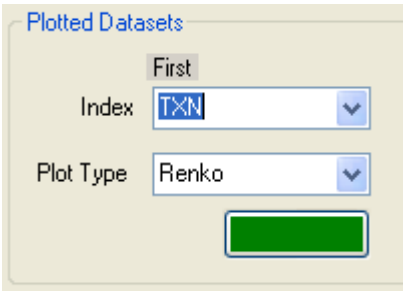
The *Average True Range* is a moving average (14-day Moving Average in this case) of the most recent and pervious 13 values. The fixed box size is set to the value of the Average True Range calculated using the most recent data. Below is the Renko chart we have been using as an example, configured for a fixed box size calculated using the Average True Range.



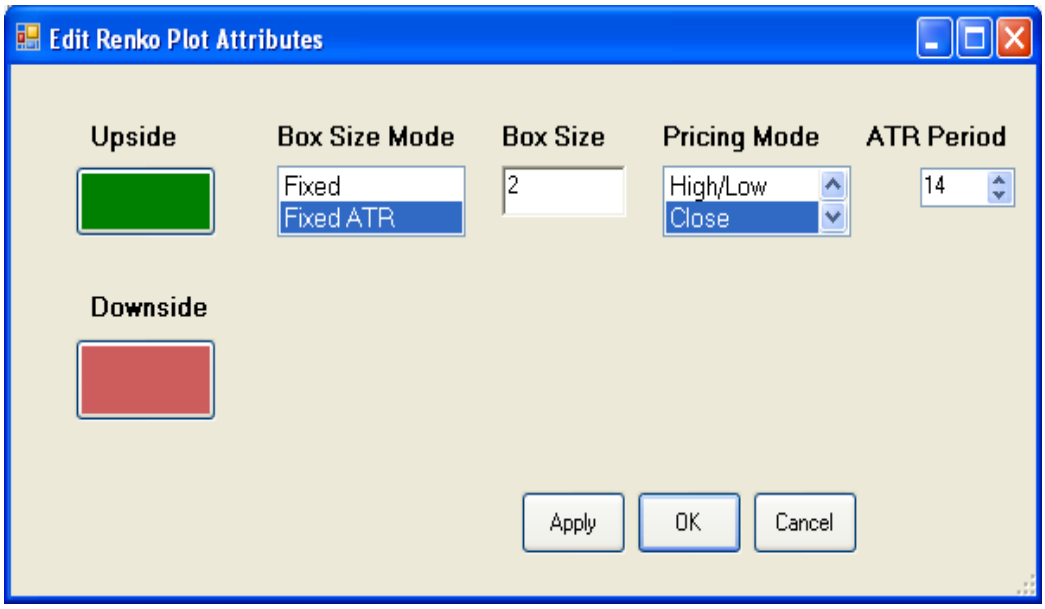
In this case the fixed box size was calculated to be approximately 0.6.

Renko Chart Dialog Box

The Renko box size mode can be set using the Attribute dialog for the chart, found in the Primary Chart Dialog page.

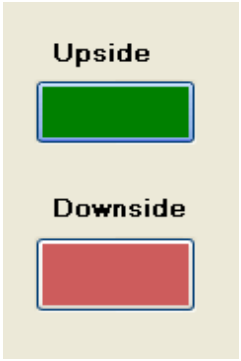


Select the Attribute box (the green box above) and the Renko dialog will appear.

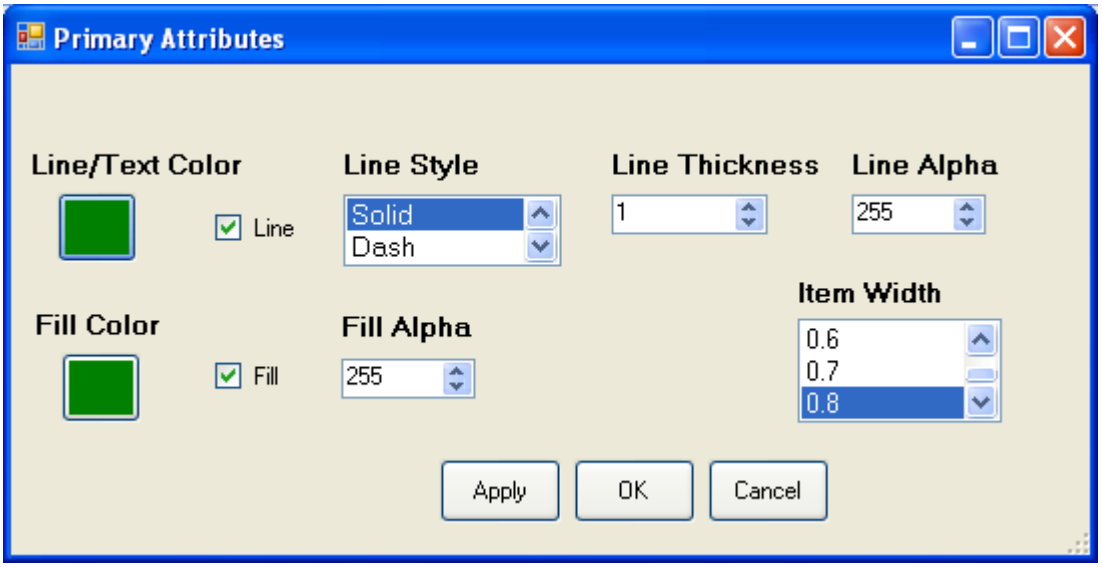


Renko Colors

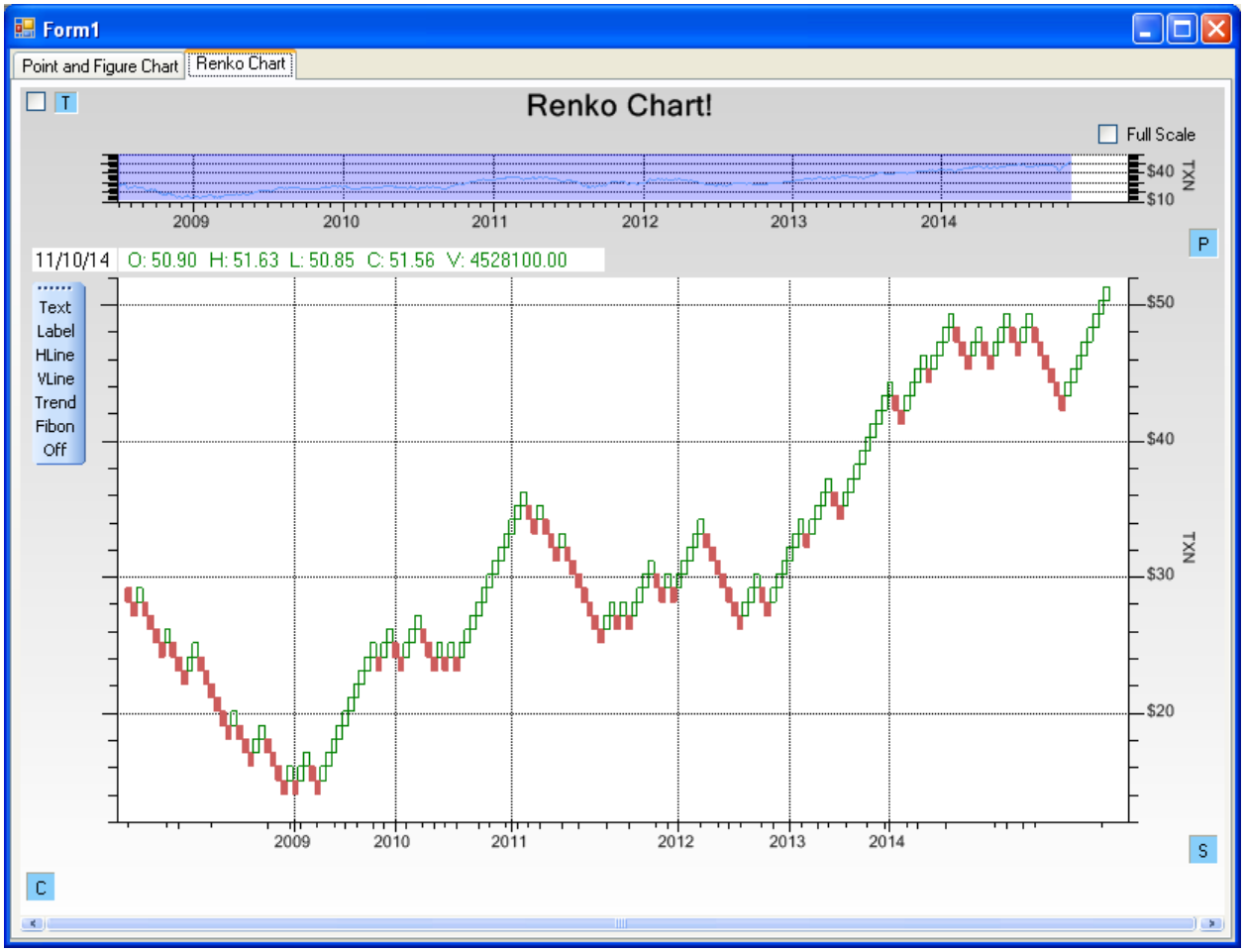
Select the Upside and/or Downside Attributes buttons to change the Renko colors.



At that point an Attributes Dialog box will appear.



If you want to use a solid box for the upside color, set the **Line/Text Color**, and the **Fill Color** to the color you want, and check both **Line** and **Fill**. The other fields should remain unchanged. You would do the same for the Downside color. The resulting chart would look like one of the two preceding charts, where the upside uses a solid green box, and the downside uses a solid red box. If you want one or both of the boxes to be unfilled, uncheck the Fill checkbox. In the example below, the **Fill** checkbox for the **Upside Color** has been unchecked.



In code, this looks like:

C#

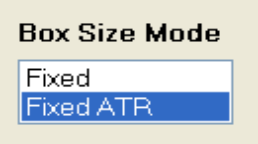
```
 FinRenkoChartPlot plotobj = this.AddPrimaryChartRenko();

// Green box outline, no fill
 plotobj.PlotUpAttribute = new ChartAttribute(Color.Green, 1, DashStyle.Solid);

// IndianRed box outline, and IndiantRed box fill
 plotobj.PlotDownAttribute = new ChartAttribute(Color.IndianRed,1,DashStyle.Solid,
Color.IndianRed);
```

VB

```
FinRenkoChartPlot plotobj = Me.AddPrimaryChartRenko()

' Green box outline, no fill
 plotobj.PlotUpAttribute = new ChartAttribute(Color.Green, 1, DashStyle.Solid)

' IndianRed box outline, and IndiantRed box fill
 plotobj.PlotDownAttribute = new ChartAttribute(Color.IndianRed,1,DashStyle.Solid,
Color.IndianRed)
```

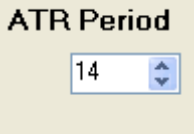
Box Size Mode

The **Box Size Mode** combo box is used to select the Renko box size mode: Fixed, or Fixed ATR.



If you select a **Box Size Mode** of Fixed, the box size is set to the value of the **Box Size** field..

If the **Box Size Mode** is set to Fixed ATR, the **Box Size** field is ignored and the ATR look-back period is specified using the **ATR Period** combo box, seen below.



If you want to set the **Box Size Mode** property under program control, set the appropriate FinRenkoChartPlot property, as returned by the AddPrimaryChartPointAndFigurecall in your program.

C#

```
FinRenkoChartPlot plotobj = this.AddPrimaryChartRenko();

plotobj.BoxSizeMode = FinChartConstants.RENKO_YSCALE_FIXED;
```

VB

```
Dim plotobj As FinRenkoChartPlot = Me.AddPrimaryChartRenko()

plotobj.BoxSizeMode = FinChartConstants.RENKO_YSCALE_FIXED
```

Specify the **BoxSizeMode** using one of the following FinChartConstants constants:

```
FinChartConstants.RENKO_YSCALE_FIXED
FinChartConstants.RENKO_YSCALE_ATRFIXED
```

If you choose the RENKO_YSCALE_FIXED box size mode, you can set the related fixed box property using the BoxSize property.

```
plotobj.BoxSize = 2;
```

If you choose the RENKO_YSCALE_ATRFIXED box size mode, you can set the related ATR look-back period using the ATRPeriod property.

```
plotobj.ATRPeriod = 14
```

Renko Price Mode

The price input to the chart takes one of three forms. You can plot the chart using Close prices only. Or you can use the most popular method which uses the days High and Low values. The third method is similar to the first option, but instead of using the Close price, the Typical Price is used, where the Typical price is defined as: (High value + Low value + Close value) / 3.

High-Low Method

Here is the basic algorithm for the High-Low method.

IF (High breaks out of the current box into a new, higher box)

Advance one column and draw the new box using the upside color and ignore the Low

ELSE (Low breaks out of the current box into a new, lower box)

Advance one column and draw the new box using the downside color

ELSE

Do nothing

The High-Low method produces more fluctuations in the resulting chart than the Close method.

Close Method

Here is the basic algorithm for the Close method

IF (Close breaks out of the current box into a new, higher box)

Advance one column and draw the new box using the upside color

ELSE (Close breaks out of the current box into a new, lower box)

Advance one column and draw the new box using the downside color

ELSE

Do nothing

Typical Price Method

The Typical Price for each OHLC event is calculated as: (High value + Low value + Close value) / 3.

IF (Typical Price breaks out of the current box into a new, higher box)

Advance one column and draw the new box using the upside color

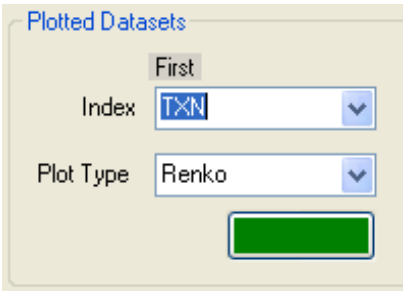
ELSE (Typical Price breaks out of the current box into a new, lower box)

Advance one column and draw the new box using the downside color

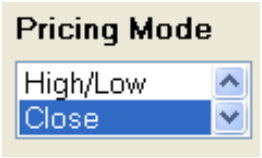
ELSE

Do nothing

The Renko Price mode can be set using the Attribute dialog for the chart, found in the Primary Chart Dialog page.



Select the Attribute box (the blue box above) and the Renko dialog will appear. Set the Pricing Mode combo box to the value you want.



If you want to set the **Pricing Mode** property under program control, set the appropriate FinRenkoChartPlot

parameter, as returned by the AddPrimaryChartRenko call in your program.

C#

```
FinRenkoChartPlot plotobj = this.AddPrimaryChartRenko();

plotobj.PricingMode = FinChartConstants.RENKO_PRICE_CLOSE;
```

VB

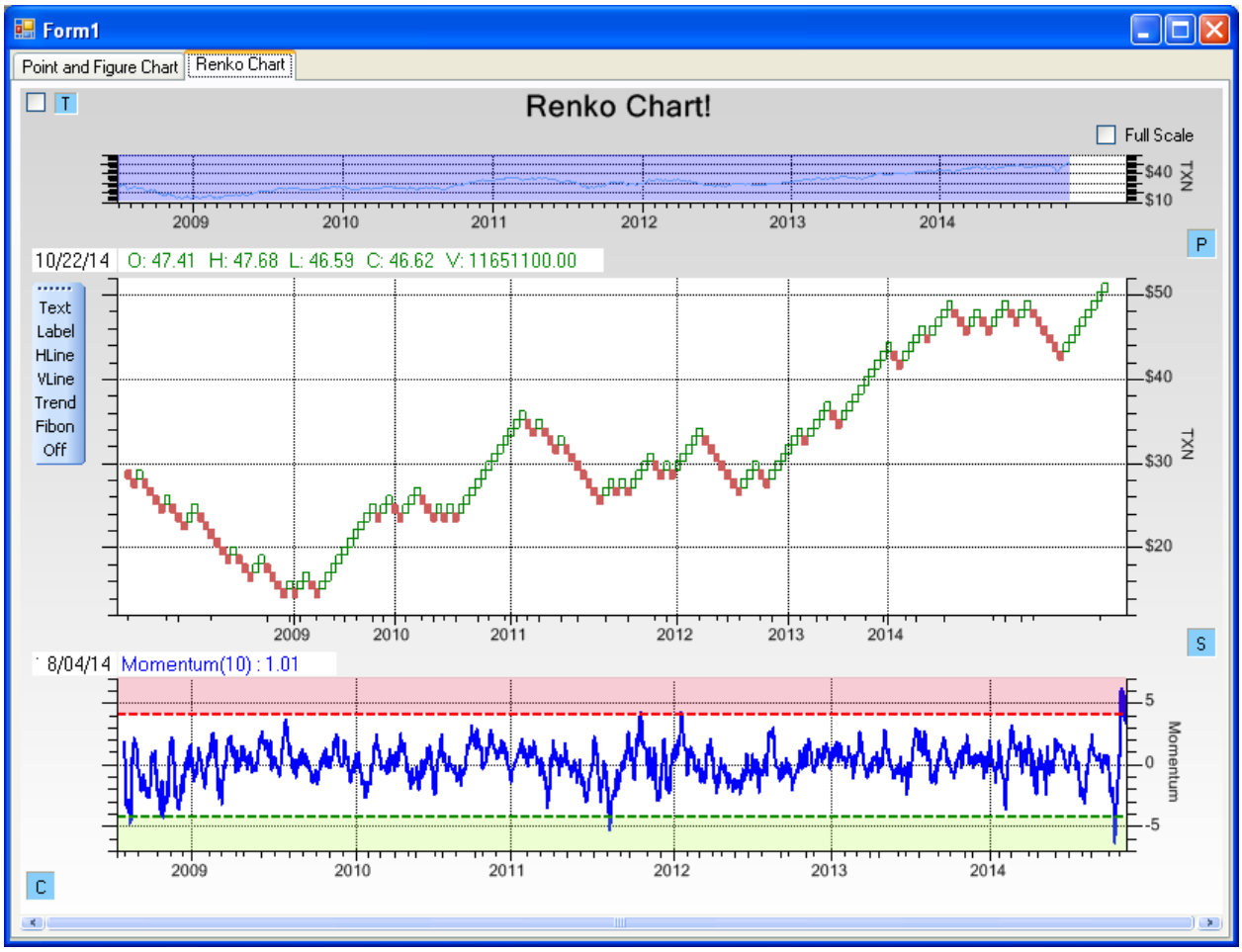
```
Dim  plotobj As FinRenkoChartPlot = Me.AddPrimaryChartRenko()

plotobj.PricingMode = FinChartConstants.RENKO_PRICE_CLOSE
```

Specify the **PricingMode** using one of the following FinChartConstants constants:

FinChartConstants.RENKO_PRICE_CLOSE
FinChartConstants.RENKO _PRICE_HIGHLOW
FinChartConstants.RENKO _PRICE_TYPICAL

Combining Renko Charts with Secondary Chart Technical Indicators

The Renko plot (and the Point and Figure plot) is a special case in the FinChartView Primary chart. This is because it uses a unique coordinate system which does not match up with the more traditional linear sequential event-based coordinate system used in the standard FinChartView Primary plot types (Line, OHLC, Candlestick, Bar, and Mountain). As you see in the examples, the coordinate system is non-linear with respect to time in the x-axis. Since its x-axis coordinate system is unique to the underlying data, it will not sync, exactly (when zooming, panning and scrolling) with the other Secondary chart technical indicator plots displayed in the FinChartView. Instead, only the starting and ending dates will sync. All other dates between the endpoints will be out of sync.



When a Renko chart, or a Point and Figure chart is displayed in the Primary chart window, the technical indicators in the secondary window(s) will only be in-sync at the endpoints.

Creating a Renko chart

The Renko charting routines are integrated in the FinChartView class. So, you create a FinChartData object with a portfolio of stocks, use that to initialize the FinChartView, and then add a Renko chart as the Primary Chart for the FinChartView, using AddPrimarcyChartRenko. See the example PointAndFigureExample.RenkoPlotUserControl1.

C#

```
public void InitializeChart()
{

   this.PreferredSize = new Size(800, 600);
   // Define the ticker symbols
   String[] idStrings = { "BA", "INTC", "IBM", "TXN", "AMAT", "CSCO" };
   String[] tickerStrings = { "BA", "INTC", "IBM",  "TXN", "AMAT", "CSCO" };

  ChartCalendar startdate = new ChartCalendar(2008, ChartObj.JULY, 2);
  ChartCalendar enddate = new ChartCalendar(); // Today

  FinYahooURLCurrentDataSource finStockData = new FinYahooURLCurrentDataSource();
  FinYahooURLHistoricalDataSource finStockHistoricalData = new
```

```
FinYahooURLHistoricalDataSource();

  for (int i = 0; i < idStrings.Length; i++)
  {
     finStockHistoricalData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
     finStockData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
  }
  finStockHistoricalData.AddCurrencyLookupItem("TXN", "#");
  FinChartData finChartData = new FinChartData(finStockHistoricalData, finStockData,
idStrings, startdate, enddate);

  // Add all of the column items to the table
  finChartData.AddAllColumnItems();
  // Initially turn the table off.
  this.EnableFinChartTable = false;

  // Assign FinChartView's data
  InitFinChartView(finChartData);

  // only needed if you are reinitalizing the charts
  this.ResetTechnicalCharts();
  // main title
  this.MainTitleString = "Renko Chart!";
  // Choose the stock for the initial chart, from the stocks entered into finChartData
  this.CurrentTickerString = "TXN";
  // Set zoom flag true

  FinRenkoChartPlot plotobj = this.AddPrimaryChartRenko();
  // Assign the up and down colors
   plotobj.PlotUpAttribute = new ChartAttribute(Color.Green, 1, DashStyle.Solid,
Color.Green);
   plotobj.PlotDownAttribute = new ChartAttribute(Color.IndianRed, 1, DashStyle.Solid,
Color.IndianRed);
   // Build the chart using the given parameters
   this.BuildChart();
}
```

VB

```
Public Sub InitializeChart()

 Me.PreferredSize = New Size(800, 600)
' Define the ticker symbols
 Dim idStrings As [String]() = {"BA","INTC","IBM","TXN","AMAT","CSCO"}
 Dim tickerStrings As [String]() = {"BA","INTC","IBM","TXN","AMAT","CSCO"}

 Dim startdate As New ChartCalendar(2008, ChartObj.JULY, 2)
 Dim enddate As New ChartCalendar()
' Today
 Dim finStockData As New FinYahooURLCurrentDataSource()
 Dim finStockHistoricalData As New FinYahooURLHistoricalDataSource()

 For i As Integer = 0 To idStrings.Length - 1
 finStockHistoricalData.AddTickerLookupItem(idStrings(i), tickerStrings(i))
 finStockData.AddTickerLookupItem(idStrings(i), tickerStrings(i))
 Next
 finStockHistoricalData.AddCurrencyLookupItem("TXN","#")
 Dim finChartData As New FinChartData(finStockHistoricalData, finStockData, idStrings,
startdate, enddate)

' Add all of the column items to the table
 finChartData.AddAllColumnItems()
' Initially turn the table off.
 Me.EnableFinChartTable = False

' Assign FinChartView's data
 InitFinChartView(finChartData)
```

```
' only needed if you are reinitalizing the charts
 Me.ResetTechnicalCharts()
' main title
 Me.MainTitleString ="Renko Chart!"
' Choose the stock for the initial chart, from the stocks entered into finChartData
 Me.CurrentTickerString ="TXN"
' Set zoom flag true

 Dim plotobj As FinRenkoChartPlot = Me.AddPrimaryChartRenko()
' Assign the up and down colors
 plotobj.PlotUpAttribute = New ChartAttribute(Color.Green, 1, DashStyle.Solid,
Color.Green)
 plotobj.PlotDownAttribute = New ChartAttribute(Color.IndianRed, 1, DashStyle.Solid,
Color.IndianRed)
' Build the chart using the given parameters
 Me.BuildChart()

End Sub
```

11. File and Printer Rendering Classes

ChartPrint
BufferedImage

com.quinn-curtis.chart2dnet.ChartView
FinChartView

Charts create using the ChartView, and FinChartView classes can be printed, and saved to an image file using the techniques described in the **QCChart2D** manual, QCChart2DNetManual.pdf. This chapter repeats that information, substituting examples extracted from the for **QCTAChart** examples.

High quality B&W and color printing is an important feature of the charting library. The resulting graph renders on the printer using the resolution of the output device, for both text and graphical elements of the chart, and does not transfer a grainy image from the computer to the printer. The **QCChart2D for .***Net* software uses the Microsoft .Net **PrintDocument** component to implement printing. Since the aspect ratio of the printed page is different from the aspect ratio of common displays, options are included that allow different modes for positioning and sizing the chart on the printed page.

The **BufferedImage** class converts a chart into a .Net **Bitmap** object, or saves the chart to a file in any of the graphics formats supported by the **System.Drawing.Imaging.ImageFormat** class. The image file is placeable in a web page or an application program.

Printing a Chart

Class ChartPrint

ChartObj
ChartPrint

The **ChartPrint** class uses the Microsoft .Net **PrintDocument** component to implement printing. The class selects, setups, and outputs a chart to a printer.

ChartPrint constructor

```
[Visual Basic]
Overloads Public Sub New( _
 ByVal component As ChartView,_
 ByVal nsizemode As Integer _
)
[C#]
public ChartPrint(
 ChartView component,
 int nsizemode
);
```

component Specifies the **ChartView** object to be printed.

nsizemode Specifies the printer mapping mode. Use one of the mapping mode constants:

PRT_MAX	Print the view so that paper is used maximally. Text prints proportional to other objects, aspect ratio is NOT maintained
PRT_EXACT	Print the view at the same size as the screen, at least as far as .Net maintains a one to one correspondence in the printing engine. The aspect ration of the view is maintained.
PRT_RECT	Print the view to the specified rectangle, specified using the SetPrintRect method and normalized coordinates.

Call the **ChartPrint.DoPrintDialog** method after creating the **ChartPrint** object. Then call the **ChartPrint.DoPrintPage** method, rendering the chart to the printer. If the **DoPrintDialog** method is not called prior to **DoPrintPage**, the **DoPrintPage** method automatically invokes the **DoPrintDialog** method. Subsequent calls to **DoPrintPage** will not invoke the **DoPrintDialog** method.

ChartPrint example **(extracted from the example program YahooDataSourceExample.Form1)**

[C#]

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Drawing.Imaging;
using com.quinncurtis.chart2dnet;
using com.quinncurtis.tachartnet;

namespace YahooDataSourceExample
{
    public partial class Form1 : Form
    {
        ChartPrint printobj = null;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void pageSetupToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (yahooDatasourceExampleUserControl11 != null)
                this.PageSetup(yahooDatasourceExampleUserControl11, sender, e);
        }

      private void printerSetupToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (yahooDatasourceExampleUserControl11 != null)
                this.PrinterSetup(yahooDatasourceExampleUserControl11, sender, e);

        }

      private void printPreviewToolStripMenuItem_Click(object sender, EventArgs e)
```

```
        {
            if (yahooDatasourceExampleUserControl11 != null)
                this.PrintPreview(yahooDatasourceExampleUserControl11, sender, e);

        }

        private void printToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (yahooDatasourceExampleUserControl11 != null)
                this.PrintPage(yahooDatasourceExampleUserControl11, sender, e);
        }

        private void saveImageToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (yahooDatasourceExampleUserControl11 != null)
                this.SaveAsFile(yahooDatasourceExampleUserControl11, sender, e);
        }

 // This routine displays a dialog box, in response to an event, that prompts
 // the user for the name and file type of the image to be saved. The file type is
 // derived from the file extension.
 // The chart represented by "this" object is saved to the file using
// the specified format.
    public void SaveAsFile(ChartView chartview, object sender, System.EventArgs e)
        {
            String filename = this.Name;
            SaveFileDialog imagefilechooser = new SaveFileDialog();
            imagefilechooser.Filter =
                "Image Files(*.BMP;*.JPG;*.GIF;*.TIFF;*.PNG)|
                   *.BMP;*.JPG;*.GIF;*.TIFF;*.PNG|All files (*.*)|*.*";
            imagefilechooser.FileName = filename;
            if (imagefilechooser.ShowDialog() == DialogResult.OK)
            {
                filename = imagefilechooser.FileName;
                FileInfo fileinformation = new FileInfo(filename);
                String fileext = fileinformation.Extension;
                fileext = fileext.ToUpper();
                ImageFormat fileimageformat;
                if (fileext == ".BMP")
                    fileimageformat = ImageFormat.Bmp;
                else if ((fileext == ".JPG") || (fileext == ".JPEG"))
                    fileimageformat = ImageFormat.Jpeg;
                else if ((fileext == ".GIF"))
                    fileimageformat = ImageFormat.Gif;
                else if ((fileext == ".TIF") || (fileext == ".TIFF"))
                    fileimageformat = ImageFormat.Tiff;
                else if ((fileext == ".PNG"))
                    fileimageformat = ImageFormat.Png;
                else
                    fileimageformat = ImageFormat.Bmp;

             BufferedImage savegraph = new BufferedImage(chartview, fileimageformat);
                savegraph.Render();
                savegraph.SaveImage(filename);
            }
        }


        // This routine invokes the chart objects PageSetupItem method
     public void PageSetup(ChartView charview, object sender, System.EventArgs e)
        {
            if (charview != null)
            {
                if (printobj == null)
                {
                    printobj = new ChartPrint(charview);
                }
                else
                    printobj.PrintChartView = charview;
                printobj.PageSetupItem(sender, e);
            }
        }
```

```
        // This routine invokes the chart objects printer setup dialog method
   public void PrinterSetup(ChartView charview, object sender, System.EventArgs e)
        {
            if (charview != null)
            {
                if (printobj == null)
                {
                    printobj = new ChartPrint(charview);
                }
                else
                    printobj.PrintChartView = charview;
                printobj.DoPrintDialog();
            }
        }

   // This routine invokes the chart objects PrintPreviewItem method
   public void PrintPreview(ChartView charview, object sender, System.EventArgs e)
        {
            if (charview != null)
            {
                if (printobj == null)
                {
                    printobj = new ChartPrint(charview);
                }
                else
                    printobj.PrintChartView = charview;
                printobj.PrintPreviewItem(sender, e);
            }
        }

// This routine prints a chart by invoking the chart objects DocPrintPage method
     public void PrintPage(ChartView charview, object sender, System.EventArgs e)
        {
            if (charview != null)
            {
                if (printobj == null)
                {
                    printobj = new ChartPrint(charview);
                    printobj.DoPrintDialog();
                }
                else
                    printobj.PrintChartView = charview;

                printobj.DocPrintPage(sender, e);
            }
        }
    }
}
```

[Visual Basic]

```
Imports System.IO
Imports System.Drawing.Imaging
Imports com.quinncurtis.chart2dnet
Imports com.quinncurtis.tachartnet

Public Class Form1

    Private printobj As ChartPrint = Nothing

    Private Sub pageSetupToolStripMenuItem_Clickx(ByVal sender As Object, ByVal e As
EventArgs)
        If YahooDatasourceExampleUserControl11 IsNot Nothing Then
            Me.PageSetup(YahooDatasourceExampleUserControl11, sender, e)
        End If
    End Sub
```

```
    ' This routine displays a dialog box, in response to an event, that prompts
    ' the user for the name and
    ' file type of the image to be saved. The file type is derived from the file
extension.
    ' The chart represented by "this" object is saved to the file using the specified
format.
    Public Sub SaveAsFile(ByVal chartview As ChartView, ByVal sender As Object, ByVal e As
System.EventArgs)
        Dim filename As [String] = Me.Name
        Dim imagefilechooser As New SaveFileDialog()
        imagefilechooser.Filter = "Image Files(*.BMP;*.JPG;*.GIF;*.TIFF;*.PNG)|
*.BMP;*.JPG;*.GIF;*.TIFF;*.PNG|All files (*.*)|*.*"
        imagefilechooser.FileName = filename
        If imagefilechooser.ShowDialog() = DialogResult.OK Then
            filename = imagefilechooser.FileName
            Dim fileinformation As New FileInfo(filename)
            Dim fileext As [String] = fileinformation.Extension
            fileext = fileext.ToUpper()
            Dim fileimageformat As ImageFormat
            If fileext = ".BMP" Then
                fileimageformat = ImageFormat.Bmp
            ElseIf (fileext = ".JPG") OrElse (fileext = ".JPEG") Then
                fileimageformat = ImageFormat.Jpeg
            ElseIf (fileext = ".GIF") Then
                fileimageformat = ImageFormat.Gif
            ElseIf (fileext = ".TIF") OrElse (fileext = ".TIFF") Then
                fileimageformat = ImageFormat.Tiff
            ElseIf (fileext = ".PNG") Then
                fileimageformat = ImageFormat.Png
            Else
                fileimageformat = ImageFormat.Bmp
            End If

            Dim savegraph As New BufferedImage(chartview, fileimageformat)
            savegraph.Render()
            savegraph.SaveImage(filename)
        End If
    End Sub


    ' This routine invokes the chart objects PageSetupItem method
    Public Sub PageSetup(ByVal charview As FinChartView, ByVal sender As Object, ByVal e
As System.EventArgs)
        If charview IsNot Nothing Then
            If printobj Is Nothing Then
                printobj = New ChartPrint(charview)
            Else
                printobj.PrintChartView = charview
            End If
            printobj.PageSetupItem(sender, e)
        End If
    End Sub

    ' This routine invokes the chart objects printer setup dialog method
    Public Sub PrinterSetup(ByVal charview As FinChartView, ByVal sender As Object, ByVal
e As System.EventArgs)
        If charview IsNot Nothing Then
            If printobj Is Nothing Then
                printobj = New ChartPrint(charview)
            Else
                printobj.PrintChartView = charview
            End If
            printobj.DoPrintDialog()
        End If
    End Sub

    ' This routine invokes the chart objects PrintPreviewItem method
    Public Sub PrintPreview(ByVal charview As FinChartView, ByVal sender As Object, ByVal
e As System.EventArgs)
        If charview IsNot Nothing Then
```

```
            If printobj Is Nothing Then
                printobj = New ChartPrint(charview)
            Else
                printobj.PrintChartView = charview
            End If
            printobj.PrintPreviewItem(sender, e)
        End If
    End Sub

    ' This routine prints a chart by invoking the chart objects DocPrintPage method
    Public Sub PrintPage(ByVal charview As FinChartView, ByVal sender As Object, ByVal e
As System.EventArgs)
        If charview IsNot Nothing Then
            If printobj Is Nothing Then
                printobj = New ChartPrint(charview)
                charview.ControlObjectsVisible = False
                printobj.DoPrintDialog()
                charview.ControlObjectsVisible = True
            Else
                printobj.PrintChartView = charview
            End If

            printobj.DocPrintPage(sender, e)
        End If
    End Sub



    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

    End Sub


    Private Sub pageSetupToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles pageSetupToolStripMenuItem.Click
        If YahooDatasourceExampleUserControl11 IsNot Nothing Then
            Me.PageSetup(YahooDatasourceExampleUserControl11, sender, e)
        End If
    End Sub


    Private Sub printerSetupToolStripMenuItem_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles printerSetupToolStripMenuItem.Click
        If YahooDatasourceExampleUserControl11 IsNot Nothing Then
            Me.PrinterSetup(YahooDatasourceExampleUserControl11, sender, e)
        End If
    End Sub

    Private Sub printPreviewToolStripMenuItem_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles printPreviewToolStripMenuItem.Click
        If YahooDatasourceExampleUserControl11 IsNot Nothing Then
            Me.PrintPreview(YahooDatasourceExampleUserControl11, sender, e)
        End If
    End Sub

    Private Sub printToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles printToolStripMenuItem.Click
        If YahooDatasourceExampleUserControl11 IsNot Nothing Then
            Me.PrintPage(YahooDatasourceExampleUserControl11, sender, e)
        End If
    End Sub

    Private Sub saveImageToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles saveImageToolStripMenuItem.Click
        If YahooDatasourceExampleUserControl11 IsNot Nothing Then
            Me.SaveAsFile(YahooDatasourceExampleUserControl11, sender, e)
        End If
    End Sub
End Class
```

Capturing the Chart as a Buffered Image

Class BufferedImage

ChartObj
BufferedImage

The **BufferedImage** class creates a **Bitmap** object that renders a **FinChartView** object into an image buffer. The rendering takes place when the **BufferedImage**.Render method or **BufferedImage.SaveImage** method is called.

BufferedImage constructor

```
[VB]
Overloads Public Sub New( _
 ByVal component As ChartView, _
 ByVal imgformat As ImageFormat _
)

[Visual Basic]
Overloads Public Sub New( _
 ByVal component As ChartView _
)

[C#]
public BufferedImage(
 ChartView component,
 ImageFormat imgformat
);


public BufferedImage(
 ChartView component
);
```

component	The **ChartView** object that is the source for the chart image.
imageformat	An image format object specifying the format of the rendered image.

The **BufferedImage.GetBufferedImage** method converts the chart to the .Net **Bitmap** object specified by the imageformat object and returns a reference the resulting bitmap.

BufferedImage example (extracted from the example program YahooDataSourceExample.Form1)

[C#]

```
// This routine displays a dialog box, in response to an event, that prompts
// the user for the name and
// file type of the image to be saved.
//  The file type is derived from the file extension.
// The chart represented by "this" object is saved to the file
// using the specified format.
public void SaveAsFile(FinChartView chartview, object sender, System.EventArgs e)
{
```

```
    String filename = this.Name;
    SaveFileDialog imagefilechooser = new SaveFileDialog();
    imagefilechooser.Filter =
"Image Files(*.BMP;*.JPG;*.GIF;*.TIFF;*.PNG)|*.BMP;*.JPG;*.GIF;*.TIFF;*.PNG|All files
(*.*)|*.*";
    imagefilechooser.FileName = filename;
    if (imagefilechooser.ShowDialog() == DialogResult.OK)
    {
        filename = imagefilechooser.FileName;
        FileInfo fileinformation = new FileInfo(filename);
        String fileext = fileinformation.Extension;
        fileext = fileext.ToUpper();
        ImageFormat fileimageformat;
        if (fileext == ".BMP" )
            fileimageformat = ImageFormat.Bmp;
        else if ((fileext == ".JPG") || (fileext == ".JPEG"))
            fileimageformat = ImageFormat.Jpeg;
        else if ((fileext == ".GIF"))
            fileimageformat = ImageFormat.Gif;
        else if ((fileext == ".TIF") || (fileext == ".TIFF"))
            fileimageformat = ImageFormat.Tiff;
        else if ((fileext == ".PNG"))
            fileimageformat = ImageFormat.Png;
        else
            fileimageformat = ImageFormat.Bmp;

        BufferedImage savegraph = new BufferedImage(chartview, fileimageformat);
        savegraph.Render();
        savegraph.SaveImage(filename);
    }
}
```

[VB]

```
' This routine displays a dialog box, in response to an event, that prompts
' the user for the name and
' file type of the image to be saved. The file type is derived from the file
'  extension.
' The chart represented by "this" object is saved to the file using the specified
' format.

Public Sub SaveAsFile(ByVal chartview As FinChartView, _
    ByVal sender As Object, ByVal e As System.EventArgs)
    Dim filename As [String] = Me.Name
    Dim imagefilechooser As New SaveFileDialog()
    imagefilechooser.Filter = _
"Image Files(*.BMP;*.JPG;*.GIF;*.TIFF;*.PNG)|*.BMP;*.JPG;*.GIF;*.TIFF;*.PNG|All files
(*.*)|*.*"
     imagefilechooser.FileName = filename
     If imagefilechooser.ShowDialog() = DialogResult.OK Then
         filename = imagefilechooser.FileName
         Dim fileinformation As New FileInfo(filename)
         Dim fileext As [String] = fileinformation.Extension
         fileext = fileext.ToUpper()
         Dim fileimageformat As ImageFormat
         If fileext = ".BMP" Then
             fileimageformat = ImageFormat.Bmp
         Else
             If fileext = ".JPG" Or fileext = ".JPEG" Then
                 fileimageformat = ImageFormat.Jpeg
             Else
                 If fileext = ".GIF" Then
                     fileimageformat = ImageFormat.Gif
                 Else
                     If fileext = ".TIF" Or fileext = ".TIFF" Then
                         fileimageformat = ImageFormat.Tiff
                     Else
                         If fileext = ".PNG" Then
                             fileimageformat = ImageFormat.Png
                         Else
```

```
                        fileimageformat = ImageFormat.Bmp
                    End If
                End If
            End If
        End If
        Dim savegraph As New BufferedImage(chartview, fileimageformat)
        savegraph.Render()
        savegraph.SaveImage(filename)
    End If
End Sub 'SaveAsFile
```

12. Regionalization for non-USA Markets

FinStrings

We have provided a structure for adjusting the software for different cultures and languages. All of the pre-defined strings in the software have been moved to the static class FinStrings, which can be modified at run-time. You can create multiple sets of strings, one for each unique region you sell to, and initialize the software to that set at run-time. While something similar is often done using resource files in the final .Net application program, it was not possible to add a user-customizable resource file to a pre-compiled library, like our QCTAChartNet dll.

Apart from the strings, there are a couple other areas which benefit from regionalization. First is the use of the "," (comma) in some locales as the decimal separator, in place of the "." (period). Our software uses the standard numeric conversion routines found in .Net, for converting numeric values to strings, and these automatically take into account the proper format for the region recognized by the computer. So, you shouldn't have to do anything there.

Also, date/time values are subject to regional differences; specifically the order of the month-day-year in short form strings of the form 10/1/2011 ("M/dd/yyyy" US English format), compared to 1/10/2011 (("dd/M/yyyy" European format). The default is the US English format ("M/dd/yyyy"). If you want to use the European format, set the FinChartView property EuroDateFormat true.

```
C#
   this.EuroDateFormat = true;

VB
   Me.EuroDateFormat = True
```

We use a .Net Dictionary object to store key/value pairs for storing the strings used in the software. This object is defined in the FinStrings class as:

```
public static  Dictionary<String, String> stringLookupTable = new Dictionary<String, String>();
```

The key for each string in the dictionary is the original English language string. The default value for each key/value pair is also the English language string. Everywhere in the software we retrieve the appropriate string value from the stringLookupTable using the English language key, but the string retrieved can be anything you want. You can change the default value associated with a key, using code similar to below:

```
FinStrings.SetStringItem("Line Width", "Ancho de línea");
```

where "Line Width" is the English language key, and "Ancho de línea" is the Dictionary value for the "Line Width" key. If you call this at the beginning of your program, all string references in the software, originally "Line Width" will be changed to "Ancho de línea".

If at any point you want to retrieve the string value associated with a key, use the FinStrings.GetStringItem method.

C#

```
String lineWidthString = FinStrings.GetStringItem("Line Width");
```

VB

```
Dim lineWidthString As String = FinStrings.GetStringItem("Line Width")
```

There is a long list of strings you can set in this manner. Many of them are associated with the current value tables (Yahoo and Quandl based) which can be displayed at the top of the FinChartView window. If you do not plan to use those tables, you don't need change those strings. Most of the other strings are associated with the dialog boxes the end-user can customize the software with.

Note that the Yahoo and Quandl table strings use English language keys which do not include spaces. The keys do not include spaces, but the actual string value does. For example

DaysHigh	Days High
DaysLow	Days Low
PreviousClose	Previous Close

Also, the values may include an embedded newline character, which forces the Yahoo and Quandl table headers to display using multiple lines. These are designated in the strings using the substring "<CR>".

ChangeFromYearLow	Change From<CR>52-Week Low
PercentChangeFromYearLow	% Change From<CR>52-Week Low
LastTradeRealtimeWithTime	Last Trade R-T<CR>With Time

If you change any of the string values, and want to include a line break, you can insert either the <CR> sub string,

C#

```
String newChangeFromYearLowString = "Change From<CR>52-Week Low";
```

VB

```
Dim newChangeFromYearLowString As String = "Change From<CR>52-Week Low"
```

or or the new line character "\n"

C#

```
String newChangeFromYearLowString =  "Change From\n52-Week Low";

FinStrings.SetStringItem("ChangeFromYearLowString" , newChangeFromYearLowString);
```

VB

```
Dim newChangeFromYearLowString As String =  "Change From\n52-Week Low"

FinStrings.SetStringItem("ChangeFromYearLowString" , newChangeFromYearLowString)
```

Key Item <String> **Value <String>**

General Chart and Dialog Box Items	
Default Chart Fontname	Microsoft Sans Serif
Default Dialog Fontname	Microsoft Sans Serif
ADX	ADX
DM+	DM+
DM-	DM-
Boll. Bands	Boll. Bands
LL	LL
UL	UL
MA	MA
%b	%b
BandWidth	BandWidth
Show Table	Show Table
T	T
S	S
Full Scale	Full Scale
P	P
O	O
H	H
L	L
C	C
V	V
OI	OI
Ticker	Ticker
Stop Program ?	Stop Program ?
Invalid Ticker Symbol	Invalid Ticker Symbol
High	High
Low	Low
Close	Close
Open Interest	Open Interest
Tool Strip	Tool Strip
Label	Label
Text	Text
HLine	HLine
VLine	VLine
Trend	Trend
Fibon	Fibon
Off	Off
On	On
Color	Color
Line Style	Line Style
Solid	Solid
Dash	Dash
DashDot	DashDot
DashDotDot	DashDotDot

Dot	Dot
Line Thickness	Line Thickness
Alpha Value	Alpha Value
Apply	Apply
OK	OK
Cancel	Cancel
Edit Attribute	Edit Attribute
Edit Attributes	Edit Attributes
Edit Table Attributes	Edit Table Attributes
Edit Fill Colors	Edit Fill Colors
Edit Limit Attributes	Edit Limit Attributes
Edit Indicator Limit Attributes	Edit Indicator Limit Attributes
Edit Renko Plot Attributes	Edit Renko Plot Attributes
Edit Point and Figure Attributes	Edit Point and Figure Attributes
Edit Line Attributes	Edit Line Attributes
Edit Line Plot Attributes	Edit Line Plot Attributes
Edit Bar Plot Attributes	Edit Bar Plot Attributes
Edit OHLC Plot Attributes	Edit OHLC Plot Attributes
Edit Candlestick Plot Attributes	Edit Candlestick Plot Attributes
Edit OHLC Bar Plot Attributes	Edit OHLC Bar Plot Attributes
Edit Scatter Plot Attributes	Edit Scatter Plot Attributes
Ticker #1 Plot Attributes	Ticker #1 Plot Attributes
Primary Chart Dialog	Primary Chart Dialog
Secondary Charts Dialog	Secondary Charts Dialog
Attributes	Attributes
Fill	Fill
Fill Alpha	Fill Alpha
Line/Text Color	Line/Text Color
Fill Color	Fill Color
Line Alpha	Line Alpha
Item Width	Item Width
Line	Line
Chart Size	Chart Size
Size	Size
h	h
w	w
x	x
Misc. Colors	Misc. Colors
Button	Button
Zoom Area	Zoom Area
Main Title	Main Title
Background Colors	Background Colors
Chart Area	Chart Area
Style	Style
Color #1	Color #1
Color #2	Color #2
Plot Area	Plot Area
Axes	Axes
X-Axes	X-Axes

Line Color	Line Color
Line Width	Line Width
Text Color	Text Color
Y-Axes	Y-Axes
Fonts	Fonts
Base Font	Base Font
Font Sizes Relative to Base Font Size	Font Sizes Relative to Base Font Size
X-Axis Labels	X-Axis Labels
Y-Axis Labels	Y-Axis Labels
Y-Axis Titles	Y-Axis Titles
Tooltips	Tooltips
Annotations	Annotations
Zoom Window	Zoom Window
Data Table	Data Table
Table Grid	Table Grid
Rows	Rows
Columns	Columns
Font	Font
Colors	Colors
Striped	Striped
Low Limit	Low Limit
High Limit	High Limit
Numeric precision	Numeric precision
Line Attributes	Line Attributes
Text Attributes	Text Attributes
Show Handles	Show Handles
Upside	Upside
Downside	Downside
Box Size Mode	Box Size Mode
Traditional	Traditional
Percentage	Percentage
Fixed	Fixed
Fixed ATR	Fixed ATR
Box Size	Box Size
Pricing Mode	Pricing Mode
High/Low	High/Low
Typical Price	Typical Price
Reversal Count	Reversal Count
ATR Period	ATR Period
Plot Type	Plot Type
Traditional (Xs and Os)	Traditional (Xs and Os)
Bars	Bars
Volume Bars	Volume Bars
MA Line	MA Line
Symbol	Symbol
Size (Pts)	Size (Pts)
Background	Background
Use Background	Use Background
Plot Area Clipping	Plot Area Clipping

Horizontal Justify	Horizontal Justify
Left	Left
Right	Right
Center	Center
Vertical Justify	Vertical Justify
Bottom	Bottom
Top	Top
Plotted Datasets	Plotted Datasets
Index	Index
First	First
Second	Second
Third	Third
Y-Scale	Y-Scale
Linear	Linear
Logarithmic	Logarithmic
Normalized	Normalized
Simple Moving Average	Simple Moving Average
Period	Period
Exponential Moving Average	Exponential Moving Average
Bollinger Bands	Bollinger Bands
Enable	Enable
Bandwidth (SD)	Bandwidth (SD)
Moving Average Bands	Moving Average Bands
Bandwidth %	Bandwidth %
Parabolic SAR	Parabolic SAR
Start Index	Start Index
Step Start	Step Start
Step Increment	Step Increment
Step Max	Step Max
Add Stock	Add Stock
Reset to Defaults	Reset to Defaults
Secondary Charts	Secondary Charts
Volume Indicators	Volume Indicators
Volume and MA	Volume and MA
Smoothing	Smoothing
Money Flow	Money Flow
Limits	Limits
Oscillators	Oscillators
Stochastic	Stochastic
Relative Strength (RSI)	Relative Strength (RSI)
Williams %R	Williams %R
MACD	MACD
Fast Period	Fast Period
Slow Period	Slow Period
Signal Period	Signal Period
Rate of Change	Rate of Change
Average Directional Change (ADX)	Average Directional Change (ADX)
Momentum	Momentum
Compressed Mode	Compressed Mode

Limit Attributes	Limit Attributes
EMA	EMA
MACD Signal	MACD Signal
Histogram	Histogram
HL	HL
ROC	ROC
RSI	RSI
SMA	SMA
Sochastic %K	Sochastic %K
Period (%K)	Period (%K)
Stoch. %K	Stoch. %K
Fast %D	Fast %D
Slow %D	Slow %D
MA Volume	MA Volume
First SMA	First SMA
Second SMA	Second SMA
First EMA	First EMA
Second EMA	Second EMA
Bands	Bands
Central Line	Central Line
Volume Indicator Attributes	Volume Indicator Attributes
Exponential MA Indicator Attributes	Exponential MA Indicator Attributes
Simple MA Indicator Attributes	Simple MA Indicator Attributes
Bollinger Bands Indicator Attributes	Bollinger Bands Indicator Attributes
Moving Average Bands Indicator Attributes	Moving Average Bands Indicator Attributes
Stochastic Indicator Attributes	Stochastic Indicator Attributes
MACD Indicator Attributes	MACD Indicator Attributes
Parabolic SAR Plot	Parabolic SAR Plot
Money Flow Indicator Attributes	Money Flow Indicator Attributes
Fast	Fast
Slow	Slow
Signal	Signal
ADX Line	ADX Line
ADX Indicator Attributes	ADX Indicator Attributes
Rate of Change Indicator Attributes	Rate of Change Indicator Attributes
Momentum Indicator Attributes	Momentum Indicator Attributes
RSI Indicator Attributes	RSI Indicator Attributes
Williams %R Indicator Attributes	Williams %R Indicator Attributes
Middle	Middle
Unable to retreive item	Unable to retreive item
using lookup symbol	using lookup symbol
Ticker data file for item	Ticker data file for item
not found at location	not found at location
Stock ID	Stock ID
Ticker Symbol	Ticker Symbol
Delete operation failed; you must have at least one stock.	Delete operation failed; you must have at least one stock.

Technical analysis toolbox	Technical analysis toolbox
Show the data table	Show the data table
Edit data table options	Edit data table options
Secondary chart technical indicators dialog	Secondary chart technical\n indicators dialog
Full scale mode on/off	Full scale mode on/off
Primary chart dialog	Primary chart dialog
Refresh	Refresh
Yahoo-Table Items	
Name	Name
Open	Open
DaysHigh	Days High
DaysLow	Days Low
PreviousClose	Previous Close
Ask	Ask
Bid	Bid
YearLow	Low 52-Week
YearHigh	High 52-Week
AskRealtime	Ask Real-Time
BidRealtime	Bid Real-Time
Volume	Volume
LastTradePriceOnly	Last Trade<CR>Price Only
LastTradeDate	Last Trade Date
LastTradeTime	Last Trade<CR>Time
DaysRange	Days Range
PERatio	P/E Ratio
PEGRatio	PEG Ratio
PercentChange	% Change
Change	Change
Change_PercentChange	Change In %
YearRange	Range 52-Week
Commission	Commission
Currency	Currency
ChangeRealtime	Change Real-Time
AfterHoursChangeRealtime	After Hours<CR>Change Real-Time
DividendShare	Dividend/Share
EarningsShare	Earnings/Share
DividendYield	Dividend Yield
MarketCapitalization	Market<CR>Capitalization
EBITDA	EBITDA
BookValue	Book Value
PriceSales	Price/Sales
PriceBook	Price/Book
ShortRatio	Short Ratio
TradeDate	Trade Date
AverageDailyVolume	Average Daily<CR>Volume

EPSEstimateCurrentYear	EPS Estimate<CR>Current Year
EPSEstimateNextYear	EPS Estimate<CR>Next Year
EPSEstimateNextQuarter	EPS Estimate<CR>Next Quarter
PriceEPSEstimateCurrentYear	Price/EPS Estimate<CR>Current Year
PriceEPSEstimateNextYear	Price/EPS Estimate<CR>Next Year
FloatShares	Float Shares
OneyrTargetPrice	One Year<CR>Target Price
OrderBookRealtime	Order Book<CR>Real-Time
MarketCapRealtime	Market Cap<CR>Real-Time
ChangeFromYearLow	Change From<CR>52-Week Low
PercentChangeFromYearLow	% Change From<CR>52-Week Low
LastTradeRealtimeWithTime	Last Trade R-T<CR>With Time
ChangePercentRealtime	Change % Real-Time
LastTradeSize	Last Trade<CR>Size
ChangeFromYearHigh	Change From<CR>52-Week High
PercebtChangeFromYearHigh	% Change From<CR>52-Week High
LastTradeWithTime	Last Trade<CR>With Time
DaysRangeRealtime	Days Range<CR>Real-Time
FiftydayMovingAverage	Moving Average<CR>50-Day
TwoHundreddayMovingAverage	Moving Average<CR>200-Day
ChangeFromTwoHundreddayMovingAverage	Change From 200-D<CR>Moving Average
PercentChangeFromTwoHundreddayMovingAverage	% Change From 200-D<CR>Moving Average
ChangeFromFiftydayMovingAverage	Change From 50-D<CR>Moving Average
PercentChangeFromFiftydayMovingAverage	% Change From 50-D<CR> Moving Average
ExDividendDate	Ex Dividend Date
DividendPayDate	Dividend Pay Date
PERatioRealtime	P/E Ratio<CR>Real-Time
TickerTrend	Ticker Trend
DaysValueChange	Days Value Change
DaysValueChangeRealtime	Days Value Change<CR>Real-Time
StockExchange	Stock Exchange
Quandl Table Items	
Date	Date
Float	Float
Beta	3-Year Regression Beta
Stdev	3-year Standard Devia-<CR>tion of Stock Price
Book_dc	Book Debt to <CR>Capital Ratio
Bv_eqty	Book Value of Equity
Bv_assets	Book Value of Assets
Capex	Capital Expenditures
Cash	Cash
Cash_fv	Cash as Percentage<CR> of Firm Value
Cash_rev	Cash as Percentage<CR> of Revenue
Cash_assets	Cash as Percentage <CR>of Total Assets
Chg_ncwc	Change in Non-Cash<CR> Working Capital
Correl	Correlation with<CR> the Market

Pe_curr	Current PE Ratio
Deprec	Depreciation
Div_yld	Dividend Yield
Div	Dividends
Ebit	Earnings Before<CR> Interest and Taxes
Ebit_1t	EBIT for<CR> Previous Period
Ebitda	EBITDA
Eff_tax	Effective Tax Rate
Eff_tax_inc	Effective Tax Rate<CR> on Income
Ev	Enterprise Value
Ev_cap	EV to Invested<CR> Capital Ratio
Ev_salestr	EV to Trailing<CR> Sales Ratio
Ev_ebit	EV to <CR>EBIT Ratio
Ev_ebitda	EV to <CR>EBITDA Ratio
Ev_sales	EV To <CR>Sales Ratio
Eps_gro_exp	Expected Growth <CR>in Earnings/Share
Rev_gro_exp	Expected Growth<CR> in Revenues
Fcff	Free Cash Flow<CR> to Firm
Firm_val	Firm Value
Fixed_tot	Ratio of Fixed Assets<CR> to Total Assets
Eps_fwd	Forward Earnings/Share
Pe_fwd	Forward PE Ratio
Eps_gro	Growth in Earnings/Share
Rev_gro	Previous Year Growth<CR> in Revenues
Hilo	Hi-Lo Risk
Insider	Insider Holdings
Inst_hold	Institutional Holdings
Intang_tot	Ratio of Intangible <CR>Assets to Total Assets
Inv_cap	Invested Capital
Mkt_cap	Market Capitalization
Mkt_de	Debt to <CR>Equity Ratio
Mkt_dc	Debt to <CR>Capital Ratio
Net_inc	Net Income
Net_marg	Net Margin
Ncwc	Non-Cash <CR>Working Capital
Ncwc_rev	Non-Cash Working Capital<CR> as Percentage of Revenues
Payout	Payout Ratio
P_bv	Price to Book <CR>Value Ratio
Pe_g	PE to Growth Ratio
Op_marg	Pre-Tax <CR>Operating Margin
P_s	Price to <CR>Sales Ratio
Reinv	Reinvestment Amount
Reinv_rate	Reinvestment Rate
Rev_last	Revenues
Roc	Return on Capital
Roe	Return on Equity
Sga	Sales General and<CR> Administration Expenses
Stock_px	Stock Price

Tot_debt	Total Debt
Trad_vol	Trading Volume
Rev_12m	Trailing 12-month<CR> Revenues
Net_inc_trail	Trailing Net Income
Pe_trail	Trailing PE Ratio
Rev_trail	Trailing Revenues
Beta_vl	Value Line Beta
Ev_bv	EV to Book <CR>Value Ratio

The FinStrings module is used to define default, and static strings, for the various QCTAChart classes, when those classes are initialized. Trying to set the FinString strings using SetStringItem after any of the charts have been instantiated will not have the desired effect. Since the charts classes are normally instantiated in the main Form file, you must change any strings before that intialization takes place. The best way to do that is to initialize the string in a static method in the main Form file. You will find an example in the TimeVariableControlCharts.Form1.cs file. Call the static method using an initialization of a static variable in the global variables section of the class. This will guarantee that the strings get initialized first. Since the FinStrings class is static, you can call it anytime. It does not need instantiation.

C#

```
   public partial class Form1 : Form
    {
        ChartPrint printobj = null;
        static bool initStringsComplete = InitStrings();

        static bool InitStrings()
        {
         // This string is used in the Primary Chart Dialog table as the group box
         // title for the y-axis scaling modes

            FinStrings.SetStringItem("Y-Scale", "Y-Axis Scale");
            return true;
        }

        public Form1()
        {
            InitializeComponent();
        }


.
.
}
```

VB

```
 public partial class Form1 : Form
{
 ChartPrint printobj = null;
 static bool initStringsComplete = InitStrings();

 static bool InitStrings()
{
// This string is used in the Primary Chart Dialog table as the group box title for the
y-axis scaling modes

 FinStrings.SetStringItem("Y-Scale","Y-Axis Scale");
 return true;
}
```

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

.
.
.

13. Using Technical Analysis Charting Tools for .Net to Create Windows Applications

The primary view class of the QCTAChart library is the FinChartView class. The FinChartView class derives from the QCChart2D ChartView class, which in turn derives from the .Net System.Windows.Forms.UserControl class. It has the properties and methods of the underlying Chart View and UserControl classes.

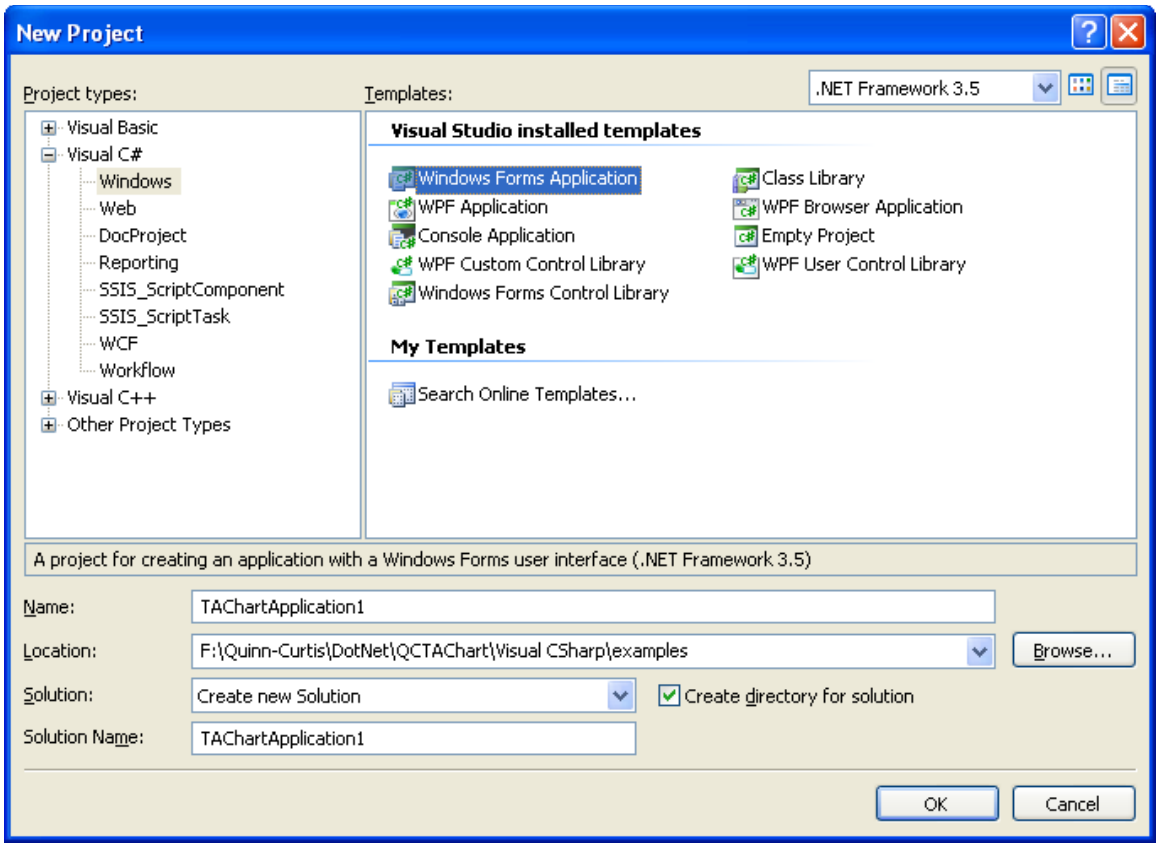
(*** Critical Note ***) Running the Example Programs

The example programs for QCTAChart software are supplied in complete source. In order to save space, they have not been pre-compiled which means that many of the intermediate object files needed to view the main form are not present. This means that FinChartView derived control will not be visible on the main Form if you attempt to view the main form before the project has been compiled. The default state for all of the example projects should be the Start Page. Before you do view any other file or form, do a build of the project. This will cause the intermediate files to be built. If you attempt to view the main Form before building the project, Visual Studio sometimes decides that the FinChartView control placed on the main form does not exist and deletes it from the project.

Follow the following steps in order to incorporate the QCTAChart classes into your program. This is not the only way to add charts to an application. In general, any technique that works with UserControl derived classes will work. We found the technique described below to be the most flexible.

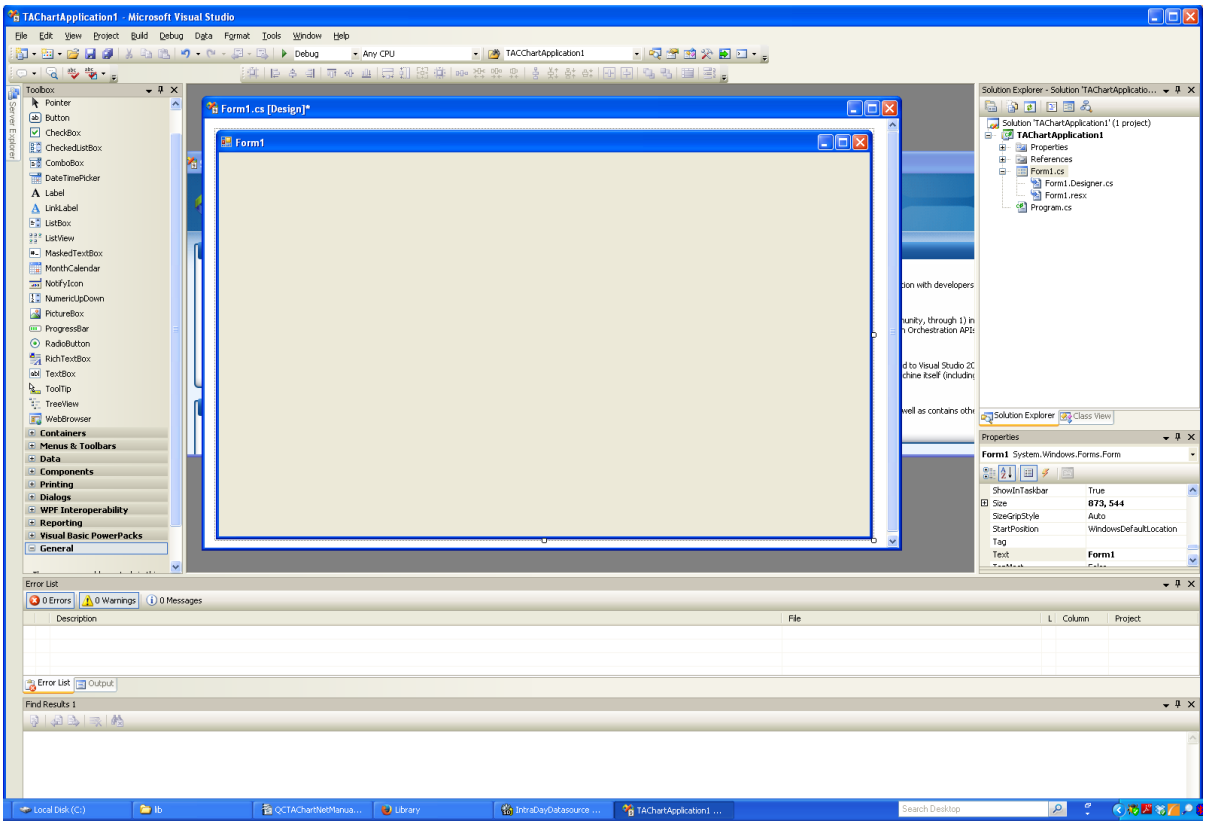
Visual C# for *.Net*

If you do not already have an application program project, create one using the Visual Studio project wizard (File | New | Project | Visual C# Projects | Windows Application). On the left, select a project type of Visual C# Projects. Give the project a unique name (our version of this example is TAChartApplication1).



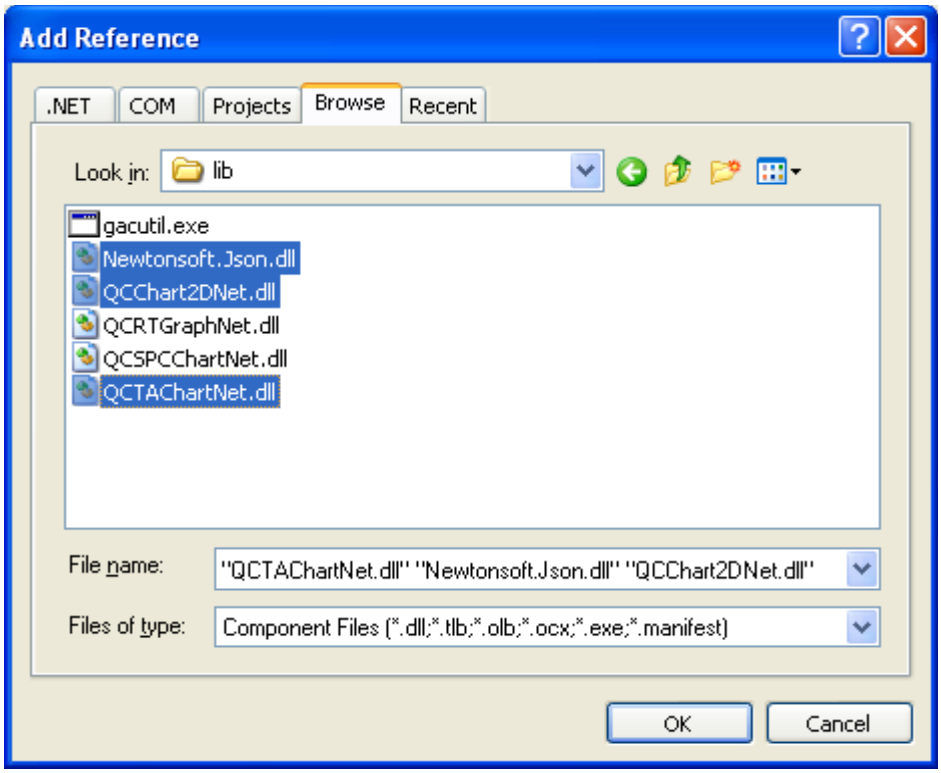
You will end with a basic Form based application. For purposes of this example, the chart is placed in the initial, default form.

- Resize Form1 to approximately the size you want your application to run in.

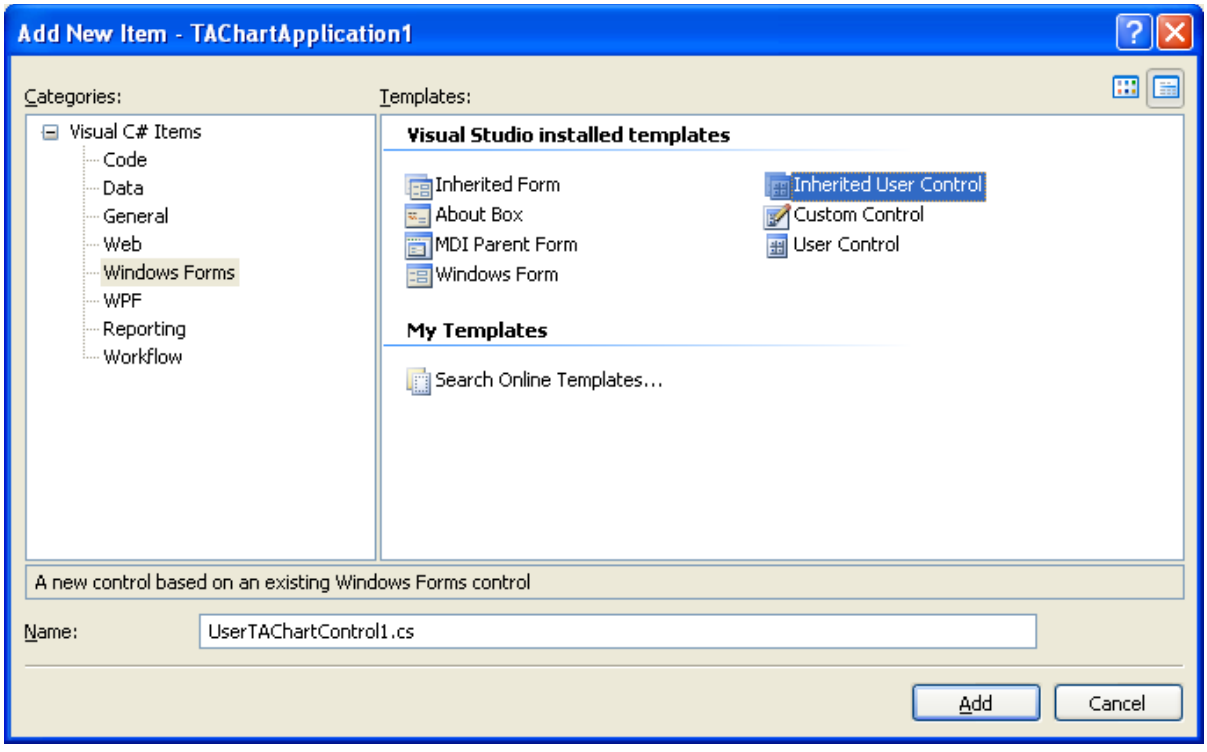


- Right click on Reference in the Solution Explorer window and select **Add Reference**. Browse to the Quinn-Curtis/DotNet/lib subdirectory and select and add the following three DLL library files: **QCTAChartNet.dll, QCChart2DNet.dll** and **Newtonsoft**.Json.dll.

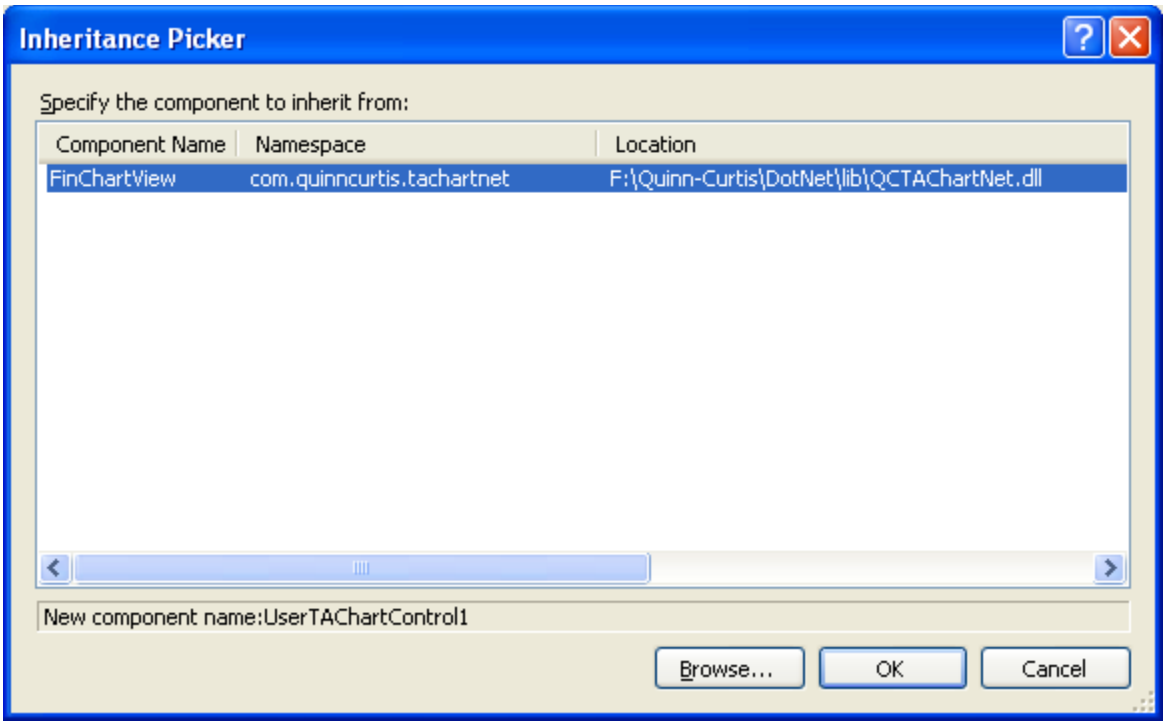
  QCTAChartNet.dll – Quinn-Curtis financial technical analysis classes
  QCChart2DNet.dll - Quinn-Curtis core charting classes
  Newtonsoft.Json.dll – Newtonsoft JSON utility classes



- Add a Inherited User Control class to the project (**Project | Add User Control**). Enter a class name of **UserTAChartControl1**.



- You will be asked to browse to a class to inherit from. Browse to the Quinn-Curtis\DotNet\lib folder and select the QCTAChartNet.dll file. The Inheritance Picker will then ask you to select the component you want to inherit from. In this case there is only one option, which is FinChartView. Then click OK and the UserTAChartControl1 class should be added to you project.



View the UserTAChartControl1.cs code.

This adds a local version of the control to the project. The C# form code should now look like:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace TAChartApplication1
{
    public partial class UserTAChartControl1 : com.quinncurtis.tachartnet.FinChartView
    {
        public UserTAChartControl1()
        {
            InitializeComponent();
        }
    }
}
```

Note that the file uses a class modifier of partial. This means that there is more code associated with the class in the related UserTAChartControl1.Designer.cs file. Normally you will not need to edit that file.

- ***Critical Step:*** Make sure you add the following lines to the top of the **UserTAChartControl1.cs** code to resolve the **QCTAChart** and other graphics classes used in the example.

```
using System.Drawing.Drawing2D;
using com.quinncurtis.chart2dnet;
using com.quinncurtis.tachartnet;
```

- Build the Solution (**Build | Build Solution**). This will compile the **UserTAChartControl1**class and make it accessible as a component on the Toolbox. If the project fails to compile, go back and check the previous steps.

  You can create as many custom chart controls as your application requires. Each custom chart control will inherit from FinChartView

- Right click on the **UserTAChartControl1** class form and view the underlying C# code. We placed all of the chart customization code in the **InitializeChart** method. Until this method is called, the **UserTAChartControl1** class appears as an empty shell. Your can call this method from the **UserTAChartControl1** class constructor;

```
public UserTAChartControl1()
{
    InitializeComponent();

    // Have the chart fill parent client area
    this.Dock = DockStyle.Fill;

    if (!IsDesignerHosted)
    {
        this.PreferredSize = new Size(800, 700);
        InitializeChart();
    }
}
```

or from somewhere outside of the class to avoid problems associated debugging errors in user controls at design time.

Add an empty InitializeChart method to the class as a placeholder. And also the IsDesignerHosted method, which we use in all of the examples to keep the control from trying to initialize when in Designer Mode.

```
public bool IsDesignerHosted
{
    get
    {
        if (LicenseManager.UsageMode == LicenseUsageMode.Designtime)
            return true;

        Control ctrl = this;
        while (ctrl != null)
        {
            if ((ctrl.Site != null) && ctrl.Site.DesignMode)
                return true;
            ctrl = ctrl.Parent;
        }
        return false;
    }
}

public void InitializeChart()
{

}
```

- Go to the main form, **Form1**. You can use either of the two following method to place the **UserTAChartControl1**class on the form.

  1. In the Form1 source file, add a variable of type **UserTAChartControl1** in the declaration section, and in the Form1 constructor, instantiate the class and add it to the Form1 controls list:

or (preferred)

2. Go to the toolbox and select the **UserTAChartControl1** from the Windows Forms list. Drop it onto the main form and size it.

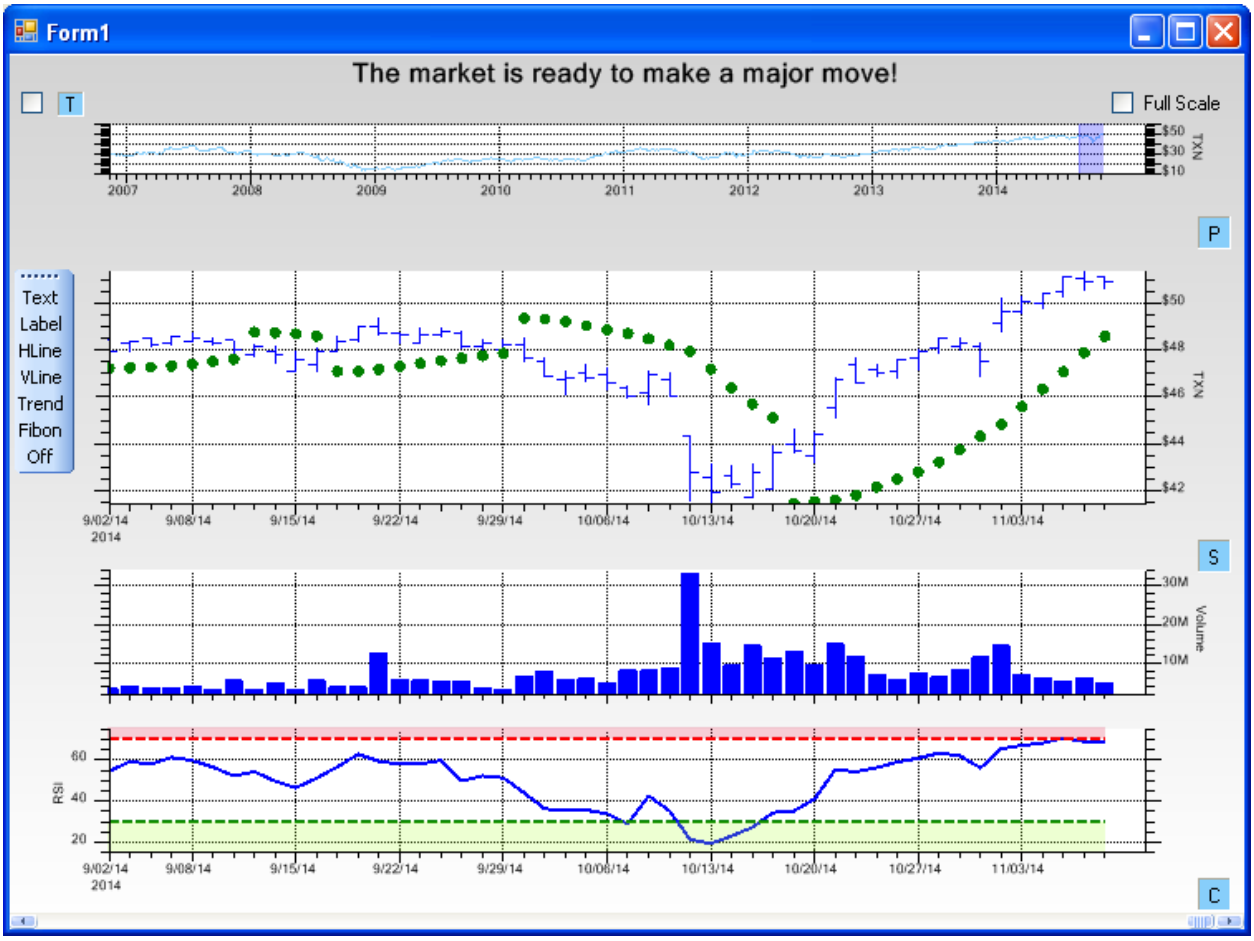
- Define the chart by customizing the **UserTAChartControl1.InitializeChart** method. See the TAChartApplication1.UserTAChartControl1.cs file for the complete code listing.

```
FinChartData finChartData = null;

  public void InitializeChart()
 {
     InitializeChartData();
     InitializeTable();
     InitializeTechnicalIndicators();
 }


public void InitializeChartData()
  {

     ChartCalendar startDate = new ChartCalendar();
     ChartCalendar stopDate = new ChartCalendar();

     String[] idStrings = {"INTC", "IBM", "TXN", "AMAT", "CSCO", "AAPL", "QQQ" };
     String[] tickerStrings = {"INTC","IBM","TXN", "AMAT", "CSCO", "AAPL", "QQQ" };
     FinYahooURLCurrentDataSource finStockData = null;
     FinYahooURLHistoricalDataSource finStockHistoricalData = null;

    // Yesterday
     stopDate.Add(ChartObj.DAY_OF_YEAR, -1);

    // starting date is 8 years ago
     startDate.Add(ChartObj.YEAR, -8);

    // Create a current data source for the table
     finStockData = new FinYahooURLCurrentDataSource();
    // Create a historical data source
     finStockHistoricalData = new FinYahooURLHistoricalDataSource();

    // Add a portfolio of stock items to the data sources
     for (int i = 0; i < idStrings.Length; i++)
     {
        finStockHistoricalData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
        finStockData.AddTickerLookupItem(idStrings[i], tickerStrings[i]);
     }
// Initialize a FinChartData object, using data sources,
// idStrings, start and ending dates.
     finChartData = new FinChartData(finStockHistoricalData, finStockData,
idStrings, startDate, stopDate);

    // Init the parent FinChartView with the FinChartData object
     InitFinChartView(finChartData);

 }



 public void InitializeTable()
 {
     // These fields are unique to the Yahoo current data source.
     // Different ones must be used if you are using
     // the Quandl current data source.

     if (finChartData != null)
     {
         // tabe does not display initially
         this.EnableFinChartTable = false;
         // 8 columns across
```

```
            this.TableDisplayColumns = 8;
            // 5 rows down
            this.TableDisplayRows = 5;
            // Use all of the available columns
            finChartData.AddAllColumnItems();

        }
    }


    public void InitializeTechnicalIndicators()
    {

        // only needed if you are reinitalizing the charts
        this.ResetTechnicalCharts();
        this.MainTitleString = "The market is ready to make a major move!";
        // Specify starting ticker, if you don't want the first ticker
        this.CurrentTickerString = "IBM";
        // Show zoom window at top
        this.FinZoomFlag = true;

        // Initialize the primary chart window
        FinPlotParameters plotobj = this.AddPrimaryChart("TXN",  ChartObj.OHLC);

        // Add parabolic SAR indicator to primary chart
        FinParabolicSARPlot parasarplot =  this.AddParabolicSARToPrimaryChart();

        // Volume secondary chart
        FinVolumePlot volumeplot = this.AddVolumeChart();
        // Add RTI indicator as secondary chart
        FinRSIIndicatorPlot rsiplot = this.AddRSIIndicatorChart();
        // Use standard (non-compressed) layout
        this.ChartLayoutMode = FinChartView.STANDARD_LAYOUT;
        // Build the chart
        this.BuildChart();

    }
```

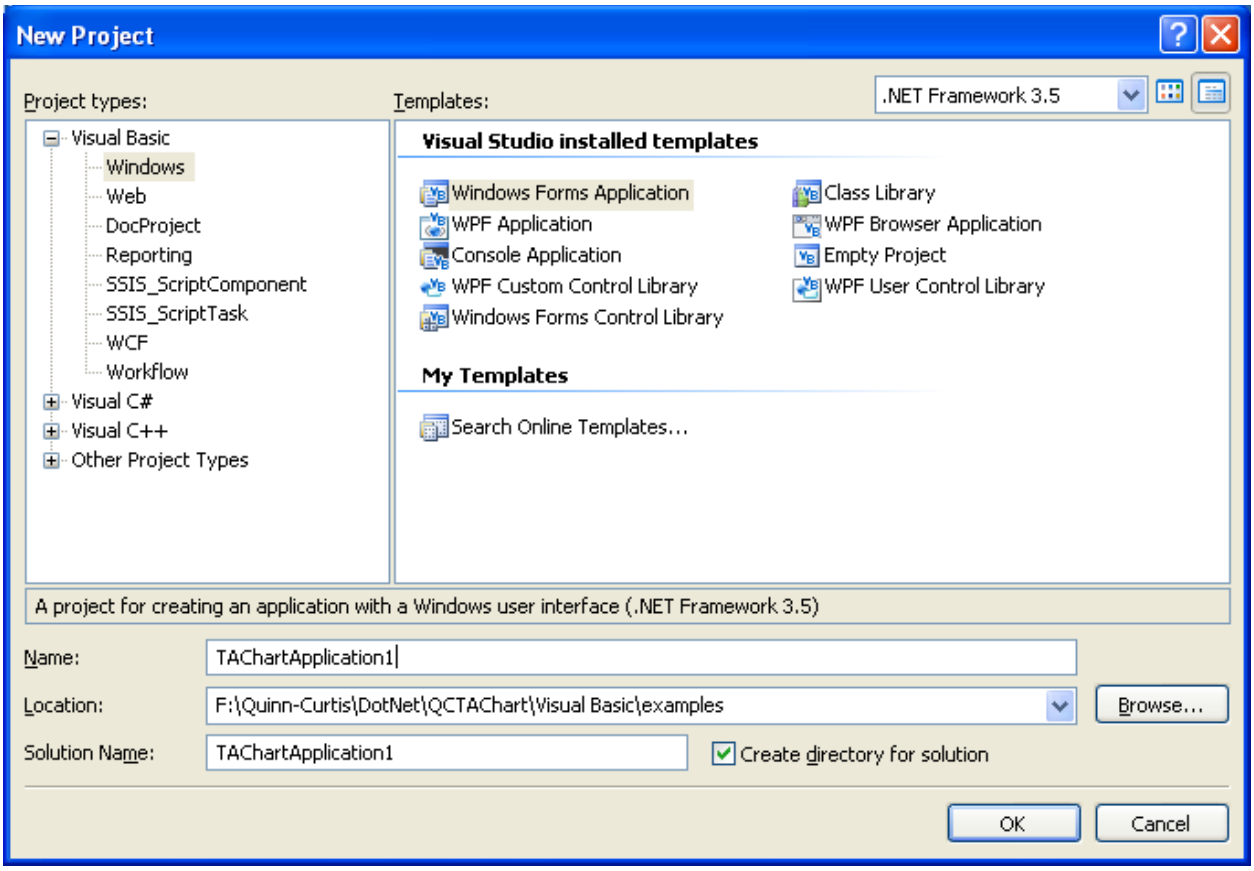
- You should be able to compile the project without error. No chart will be visible yet.

- You should now be able to compile, run and view the entire project. Any changes you make in the **UserTAChartControl1** form will be reflected in the application. If you still have problems go back and study the many example programs we have provided.

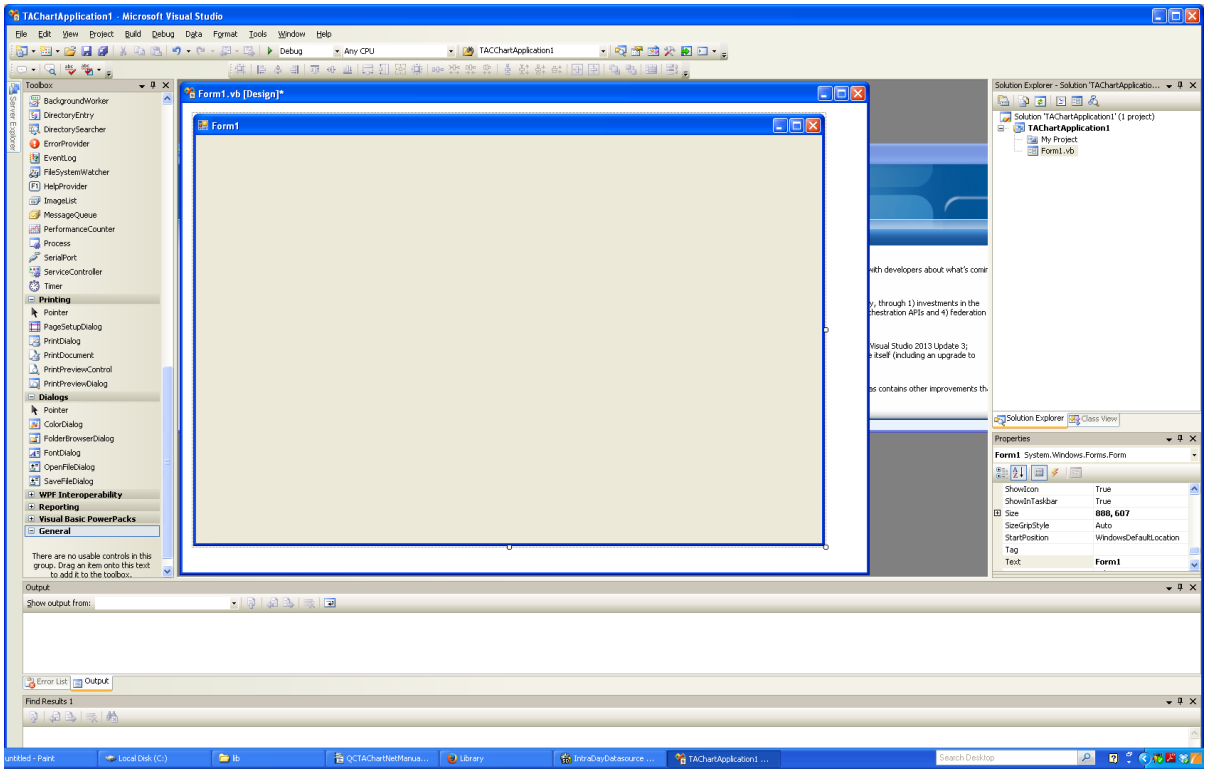


Visual Basic for .*Net*

If you do not already have an application program project, create one using the Visual Studio project wizard (File | New | Project | Visual C# Projects | Windows Application). On the left, select a project type of Visual Basic Projects. Give the project a unique name (our version of this example is TAChartApplication1).

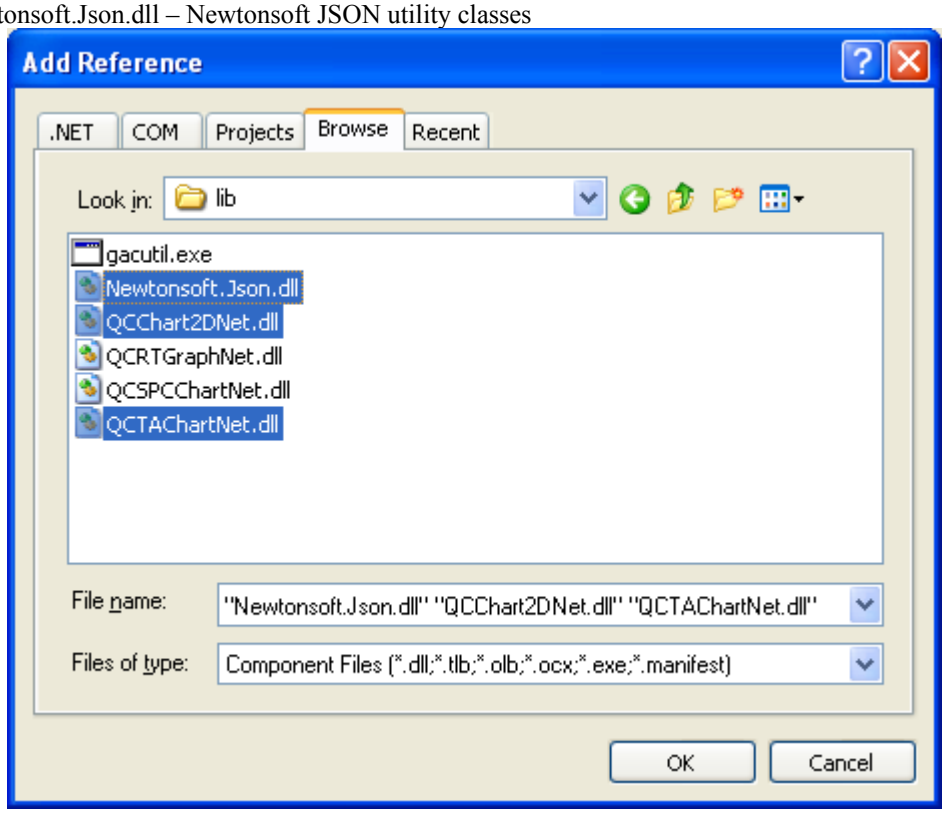


- Resize Form1 to approximately the size you expect your application to be.

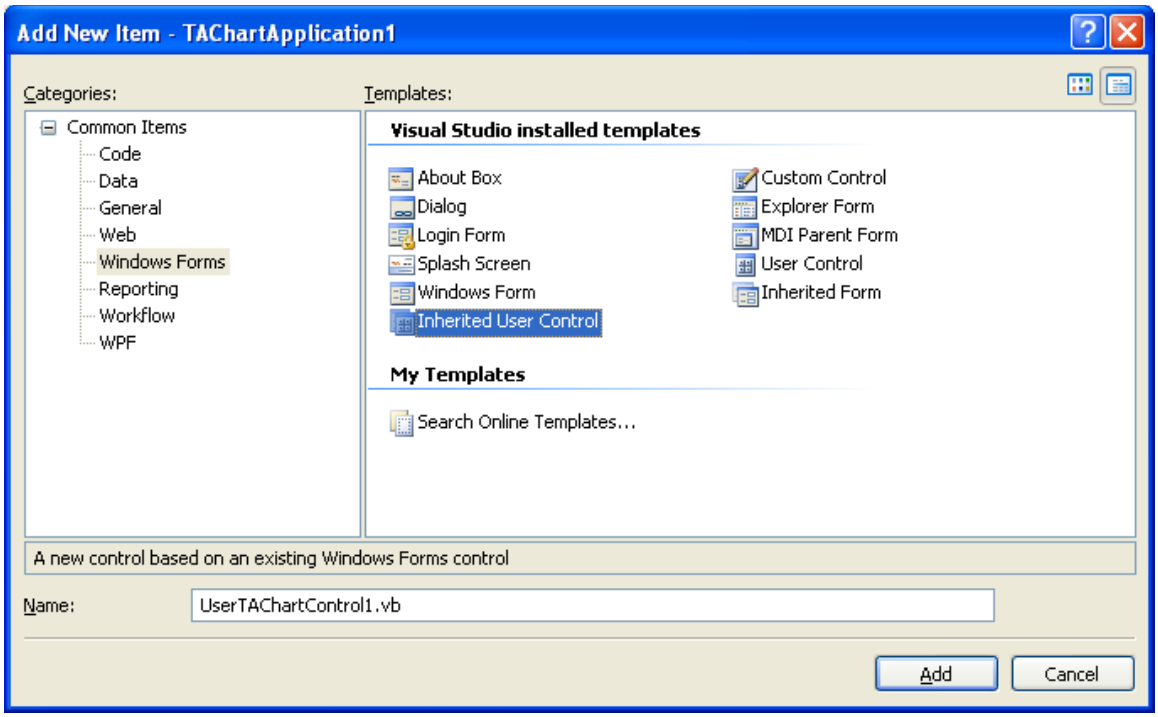


- In the Solution, right-click on TAChartApplication1 and select Add Reference. Browse to the Quinn-Curtis\DotNet\lib folder and add the DLL library files: **QCChart2DNet.dll, QCTAChartNet.dll** and **Newtonsoft.Json.dll**.

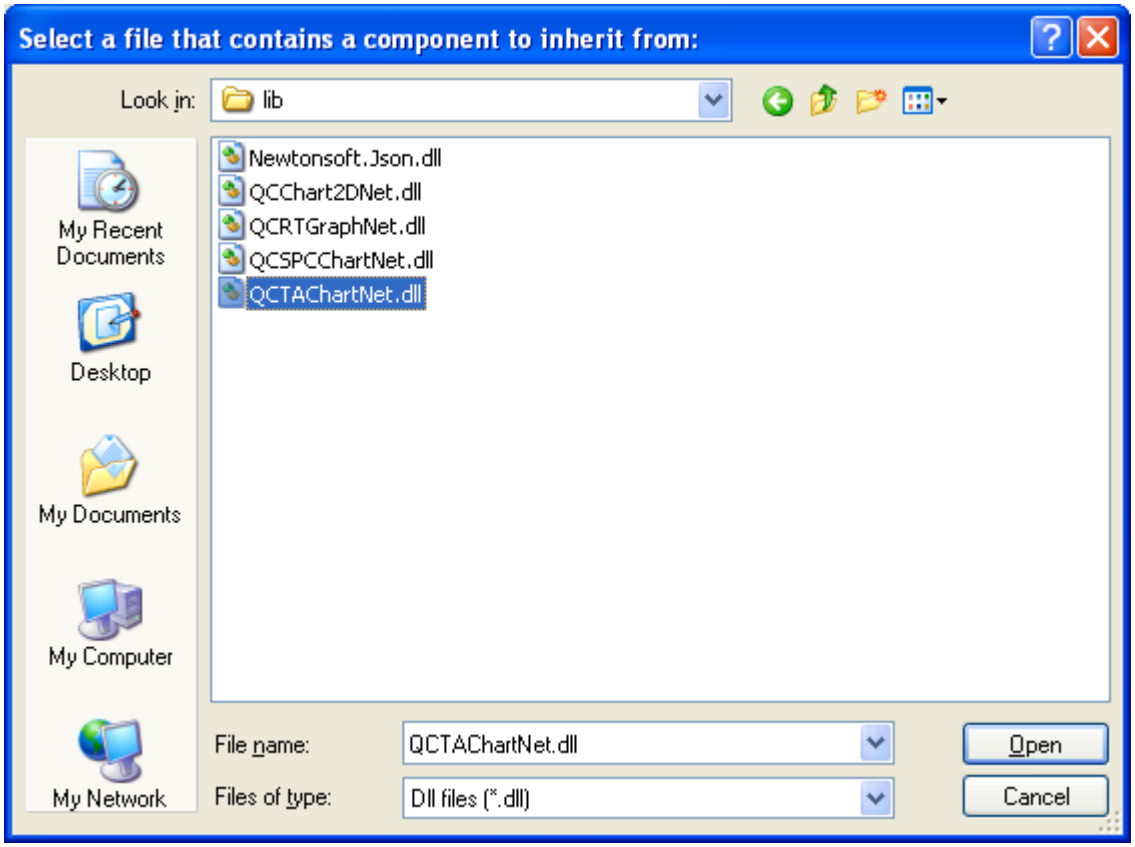
  QCTAChartNet.dll – Quinn-Curtis financial technical analysis classes
  QCChart2DNet.dll - Quinn-Curtis core charting classes
  Newtonsoft.Json.dll – Newtonsoft JSON utility classes



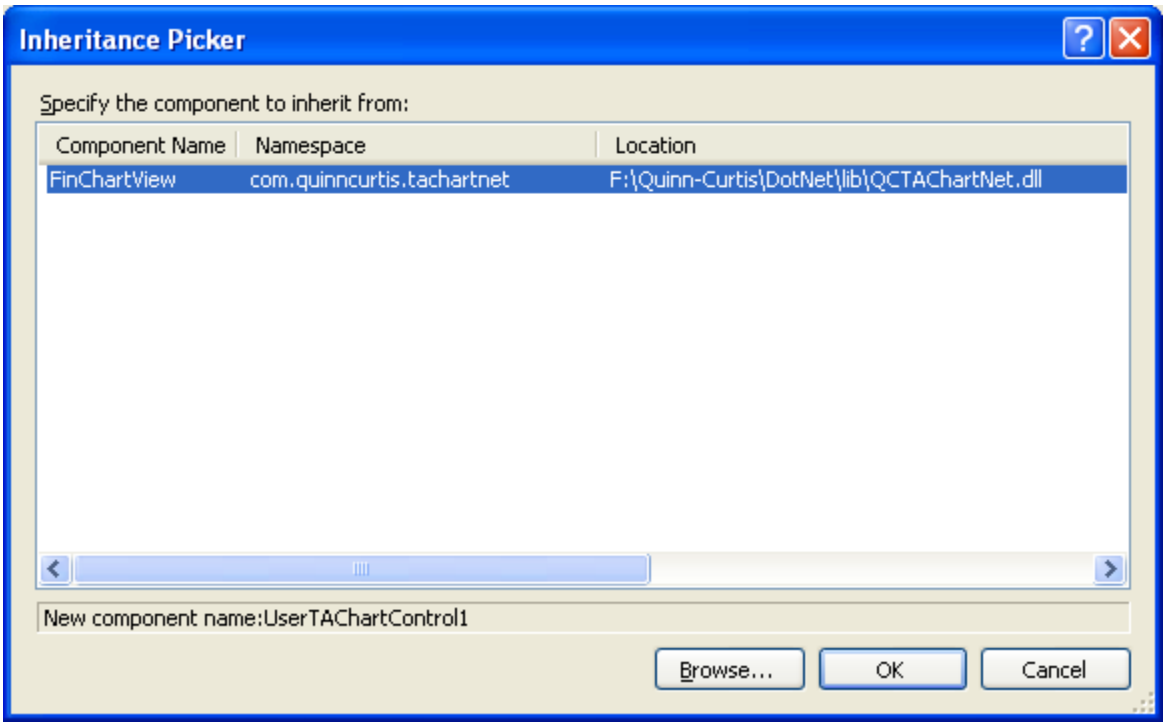
- Add a User Control class to the project (**Project | Add User Control**). Enter a class name of **UserTAChartControl1**. Select the template Inherited User Control. We use *the Inherited User Control* because it makes it easier to specify the FinChartView as the base class of the user control. The VB Inherits clause for a inherited user control shows up in the normally hidden UserTAChartControl1.Designer.vb file.



- When you click Add, you will see the Inheritance Picker. Select Browse, browse to the Quinn-Curtis\DotNet\lib folder, and select the QCTAChart.DLL file and select Open.



- Select the **FinChartView** component which should be the only one listed.



- Select OK. This will create a class named **UserTAChartControl1**, derived from **FinChartView** , and add it to the project. At this time the only code in the UserTAChartControl1.vb main class file is:

```
Public Class UserTAChartControl1

End Class
```

- ***Critical Step:*** Make sure you add the following lines to the top of the UserTAChartControl1.vb code to resolve the **QCChart2D**, **QCTAChart** and other graphics classes used in the example.

```
Imports System.ComponentModel
Imports com.quinncurtis.chart2dnet
Imports com.quinncurtis.tachartnet
```

- Build the Solution (**Build | Build Solution**). This will compile the **UserTAChartControl1** class and make it accessible as a component on the Toolbox. If the project fails to compile you need to go back and check the previous steps.

  You can create as many User Controls as your application requires. Each custom chart control will inherit from the com.quinncurtis.tachartnet.FinChartView control. Or

- (Optional) You can create inherited controls from the **UserTAChartControl1** class that you already created. Create an inherited control by selecting **Project | Add Inherited Control**. Give the inherited control a unique name, i.e. **UserChartInheritedControl1**. When you select Open, choose **UserTAChartControl1** in the Inheritance Picker. The result is new control added to the project. Build the solution and the **UserChartInheritedControl1** control is added to the Toolbox in addition to the

UserTAChartControl1.

- Look at the **UserTAChartControl1** class. The chart is created in the **InitializeChart** method. Until this method is called, the **UserTAChartControl1** appears as an empty shell. Sometimes it helps to call this method from somewhere outside of the class to avoid problems associated debugging errors in user controls at design time. Click on the UserTAChartControl1 form in design mode and a load event will be added to the code. You can add the InitializeChart call there. We have also added the **IsDesignerHosted** method to prevent the chart from initializing if the control is in Designer Mode, as opposed to Runtime Mode.

```
    Public ReadOnly Property IsDesignerHosted() As Boolean
        Get
            If LicenseManager.UsageMode = LicenseUsageMode.Designtime Then
                Return True
            End If

            Dim ctrl As Control = Me
            While ctrl IsNot Nothing
                If (ctrl.Site IsNot Nothing) AndAlso ctrl.Site.DesignMode Then
                    Return True
                End If
                ctrl = ctrl.Parent
            End While
            Return False
        End Get
    End Property

    Private Sub UserTAChartControl1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        ' Have the chart fill parent client area
        Me.Dock = DockStyle.Fill

        If Not IsDesignerHosted Then
            Me.PreferredSize = New Size(800, 700)
            InitializeChart()
        End If
    End Sub

    Public Sub InitializeChart()

    End Sub
```

Go to the main form, **Form1**. Go to the toolbox and select the **UserTAChartControl1** from the Windows Forms list. Drop it onto the main form and size it. This will automatically create an instance of the **UserTAChartControl1** class and initialize it.

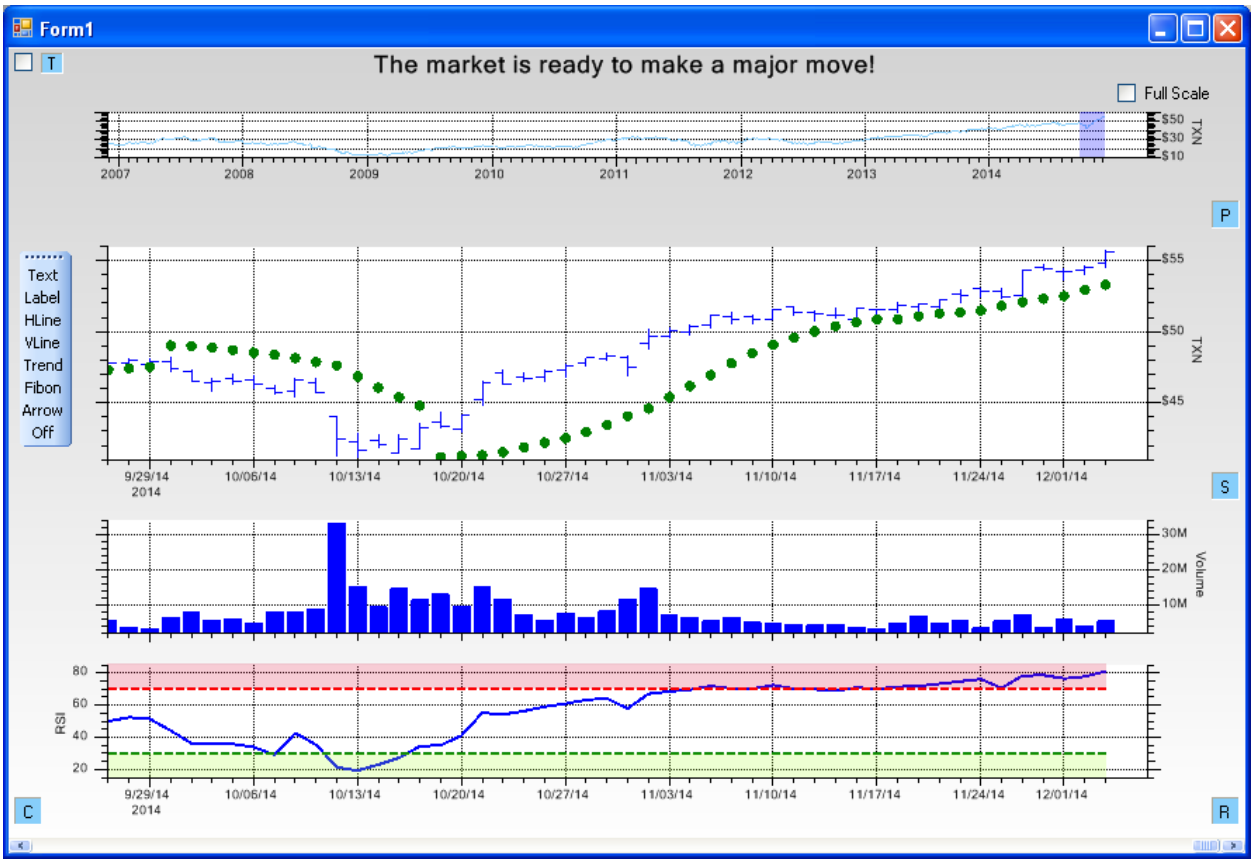
- Define the chart by customizing the UserTAChartControl1.**InitializeChart** method. See the actual TAChartApplication1.UserTAChartControl1.vb file for all of the actual code.

```
 Public Sub InitializeChart()
        InitializeChartData()
        InitializeTable()
        InitializeTechnicalIndicators()
    End Sub



    Public Sub InitializeChartData()

        Dim startDate As New ChartCalendar()
        Dim stopDate As New ChartCalendar()

        Dim idStrings As [String]() = {"INTC", "IBM", "TXN", "AMAT", "CSCO", "AAPL", _
         "QQQ"}
        Dim tickerStrings As [String]() = {"INTC", "IBM", "TXN", "AMAT", "CSCO", "AAPL", _
         "QQQ"}
        Dim finStockData As FinYahooURLCurrentDataSource = Nothing
        Dim finStockHistoricalData As FinYahooURLHistoricalDataSource = Nothing
```

```
        ' Yesterday
        stopDate.Add(ChartObj.DAY_OF_YEAR, -1)

        ' starting date is 8 years ago
        startDate.Add(ChartObj.YEAR, -8)

        ' Create a current data source for the table
        finStockData = New FinYahooURLCurrentDataSource()
        ' Create a historical data source
        finStockHistoricalData = New FinYahooURLHistoricalDataSource()

        ' Add a portfolio of stock items to the data sources
        For i As Integer = 0 To idStrings.Length - 1
            finStockHistoricalData.AddTickerLookupItem(idStrings(i), tickerStrings(i))
            finStockData.AddTickerLookupItem(idStrings(i), tickerStrings(i))
        Next
' Initialize a FinChartData object, using data sources, idStrings, start and ending dates.
        finChartData = New FinChartData(finStockHistoricalData, finStockData, idStrings,
startDate, stopDate)

        ' Init the parent FinChartView with the FinChartData object
        InitFinChartView(finChartData)

    End Sub



    Public Sub InitializeTable()
' These fields are unique to the Yahoo current data source.
' Different ones must be used if you are using
        ' the Quandl current data source.
        If finChartData IsNot Nothing Then
            ' tabe does not display initially
            Me.EnableFinChartTable = False
            ' 8 columns across
            Me.TableDisplayColumns = 8
            ' 5 rows down
            Me.TableDisplayRows = 5
            ' Use all of the available columns

            finChartData.AddAllColumnItems()
        End If
    End Sub


    Public Sub InitializeTechnicalIndicators()

        ' only needed if you are reinitalizing the charts
        Me.ResetTechnicalCharts()
        Me.MainTitleString = "The market is ready to make a major move!"
        ' Specify starting ticker, if you don't want the first ticker
        Me.CurrentTickerString = "IBM"
        ' Show zoom window at top
        Me.FinZoomFlag = True

        ' Initialize the primary chart window
        Dim plotobj As FinPlotParameters = Me.AddPrimaryChart("TXN",  ChartObj.OHLC)

        ' Add parabolic SAR indicator to primary chart
        Dim parasarplot As FinParabolicSARPlot = Me.AddParabolicSARToPrimaryChart()

        ' Volume secondary chart
        Dim volumeplot As FinVolumePlot = Me.AddVolumeChart()
        ' Add RTI indicator as secondary chart
        Dim rsiplot As FinRSIIndicatorPlot = Me.AddRSIIndicatorChart()
        ' Use standard (non-compressed) layout
        Me.ChartLayoutMode = FinChartConstants.STANDARD_LAYOUT

        ' Build the chart
        Me.BuildChart()
```

```
End Sub
```

- You should be able to compile the project without error.
- You should now be able to compile, run and view the entire project. Any changes you make in the **UserTAChartControl1** form is reflected in the application. If you still have problems go back and study the many example programs we have provided.



Index

Arrows
arrows....4, 14, 60, 67, 95
AutoScale
AutoScale....85, 113
Average Directional Indicator
Average Directional Indicator…4, 14, 44, 163, 164
Average True Range
Average True Range….42, 147, 226, 227, 244, 245
Axis
Axis.....69, 81, 85, 86, 87, 88, 89, 90, 91, 92, 108, 109, 110, 112, 113
AxisLabels
AxisLabels....91, 92, 113
AxisTitle
AxisTitle....108, 113
Background
Background....69, 81, 86, 114
BarDatapointValue
BarDatapointValue....113
Bollinger Bands
Bollinger Bands…..3, 13, 15, 40, 41, 71, 143, 146, 157, 163, 274, 275
Box Size Mode
Box Size Mode. .78, 153, 154, 226, 227, 230, 231, 244, 245, 248, 249, 273
BufferedImage
BufferedImage. 111, 113, 257, 259, 261, 263, 264, 265
Candlestick
Candlestick 2, 3, 13, 15, 16, 17, 21, 22, 59, 70, 71, 75, 93, 94, 95, 96, 98, 113, 129, 141, 150, 151, 154, 237, 251, 272
Candlestick, Candlestick....150, 154 151, 154
CartesianCoordinates
CartesianCoordinates....83, 84, 113
ChartAttribute
ChartAttribute....84, 85, 86, 113
ChartCalendar
ChartCalendar....82, 85, 89, 109, 111, 113
ChartImage
ChartImage....109, 114
ChartLabel
ChartLabel....108, 109, 113
ChartPlot
ChartPlot....93, 94, 113
ChartPrint
ChartPrint.111, 113, 257, 258, 259, 260, 261, 262, 279
printing....111, 257, 258
ChartPrint....257
ChartScale
ChartScale....83, 84, 113
ChartShape
ChartShape....109, 114
ChartSymbol
ChartSymbol....109, 114
ChartText
ChartText....108, 113
ChartTitle
ChartTitle....108, 113
ChartView
ChartView8, 82, 85, 110, 111, 113, 257, 263, 264, 265, 283
Classic Open-high-low-close
Classic Open-high-low-close....3, 13, 15
Create Windows Applications
Create Windows Applications....iv, 9, 283
CSV
CSV....111, 113
current financial data table
current financial data table....131, 163
Customer Support
customer support....iv, 9
data cursors
data cursors....69, 81, 109
data m....3, 14, 60, 195
data markers....61
DataCursor
DataCursor....110, 113
Dataset
ChartDataset....82, 93, 113
DataToolTip
DataToolTip....110, 113
Developer License
Developer License....ii
Equi-volume candlestick plot
Equi-volume candlestick plot....3, 13, 15, 21
event-based coordinate system
event-based coordinate system 14, 59, 74, 237, 251
Exponential moving averages
exponential moving averages……3, 13, 15, 36, 156
Fibonacci overlay
Fibonacci overlay…….4, 9, 14, 63, 64, 208, 209
FinADXIndicatorPlot
FinADXIndicatorPlot....70, 71, 165, 166
Financial Chart objects
Financial Chart Objects……3, iv, 9, 14, 60, 75, 195
Financial Data Sources
financial data sources....iv, 5, 133
FinArrow
FinArrow…..60, 67, 75, 76, 78, 212, 218, 220, 221
FinBollingerBandsPlot
FinBollingerBandsPlot....70, 71, 146
FinChartView
FinChartView 8, 26, 68, 70, 75, 76, 124, 125, 126, 127, 128, 131, 132, 141, 142, 163, 164, 198, 202, 206, 210, 214, 216, 217, 220, 237, 238, 239, 243, 251, 252, 253, 257, 261, 262, 263, 264, 269, 270, 283, 287, 288, 289, 290, 293, 295, 297
FinDataSourceBase

FinDataSourceBase.....72, 115
FindObj.....
FindObj.....110, 113
FinExponentialMovingAveragePlot.....
FinExponentialMovingAveragePlot.....70, 71, 144
FinFibonacciPlot.....
FinFibonacciPlot.....60, 75, 195, 210, 211, 212 75
FinGenericCurrentDataSource.....
FinGenericCurrentDataSource.....72, 73, 115
FinGenericHistoricalDataSource.....
FinGenericHistoricalDataSource.....72, 115
FinGoogleCSVFileHistoricalDataSource .....
FinGoogleCSVFileHistoricalDataSource ...72, 73, 115, 127, 128
FinGoogleHistoricalDataSource.....
FinGoogleHistoricalDataSource.....72, 115
FinGoogleURLHistoricalDataSource.....
FinGoogleURLHistoricalDataSource72, 115, 116, 117, 118
FinGoogleURLIntradayDataSource.....
FinGoogleURLIntradayDataSource….72, 73, 115, 116
FinHLine.....
FinHLine....60, 75, 76, 78, 195, 202, 203, 204, 207
FinLabel.....
FinLabel60, 65, 66, 67, 75, 76, 195, 212, 215, 216, 217
FinMABandsPlot.....
FinMABandsPlot.....70, 71, 145
FinMACDIndicatorPlot.....
FinMACDIndicatorPlot.....70, 71, 180, 181
FinMetaStockCSVFileHistoricalDataSource.....
FinMetaStockCSVFileHistoricalDataSource….72, 73, 115, 116, 121
FinMetaStockHistoricalDataSource.....
FinMetaStockHistoricalDataSource.....72, 73, 115
FinMetaStockURLHistoricalDataSource.....
FinMetaStockURLHistoricalDataSource….72, 115
FinMomentumIndicatorPlot.....
FinMomentumIndicatorPlot. .70, 71, 167, 168, 169
FinMoneyFlowIndicatorPlot.....
FinMoneyFlowIndicatorPlot. 70, 71, 169, 170, 171
FinParabolicSARPlot.....
FinParabolicSARPlot.....70, 71, 147, 290, 297
FinPointAndFigureChartPlot.....
FinPointAndFigureChartPlot70, 71, 229, 230, 231, 234, 235, 236, 238, 239
FinQuandlCSVFileHistoricalDataSource.....
FinQuandlCSVFileHistoricalDataSource....72, 73, 115, 116, 125, 126
FinQuandlCurrentDataSource.....
FinQuandlCurrentDataSource.....72, 73, 115
FinQuandlHistoricalDataSource.....
FinQuandlHistoricalDataSource.....72, 73, 115
FinQuandlURLCurrentDataDataSource.....
FinQuandlURLCurrentDataDataSource.....72, 73, 115, 131
FinQuandlURLHistoricalDataSource.....
FinQuandlURLHistoricalDataSource..72, 73, 115, 116, 118, 119, 120, 136, 137
FinRateOfChangeIndicatorPlot.....
FinRateOfChangeIndicatorPlot.....70, 71, 172
FinRenkoChartPlot.....
FinRenkoChartPlot.....70, 71, 248, 249, 250, 251, 253, 254
FinRSIIndicatorPlot.....
FinRSIIndicatorPlot.....70, 71, 173, 174, 290, 297
FinSimpleMovingAveragePlot.....
FinSimpleMovingAveragePlot.....70, 71, 144
FinStochasticIndicatorPlot.....
FinStochasticIndicatorPlot.....70, 71, 175, 176, 177
FinStrings.....
FinStrings.....76, 269, 270, 279
FinText.....
FinText60, 64, 66, 75, 76, 195, 212, 213, 214, 215, 216, 220
FinTrendLine.....
FinTrendLine.....60, 75, 76, 78, 195, 198, 199, 200
FinVLine.....
FinVLine....60, 75, 76, 78, 195, 206, 207, 208, 215
FinVolumeAndMAPlot.....
FinVolumeAndMAPlot.....70, 71, 183
FinVolumePlot.....
FinVolumePlot.....70, 71, 183, 290, 297
FinWilliamsRIndicatorPlot.....
FinWilliamsRIndicatorPlot.....70, 71, 178, 179
FinYahooCSVFileHistoricalDataSource.....
FinYahooCSVFileHistoricalDataSource.....72, 73, 115, 116, 124, 125
FinYahooCurrentDataSource.....
FinYahooCurrentDataSource.....72, 73, 115
FinYahooHistoricalDataSource.....
FinYahooHistoricalDataSource.....72, 73, 115
FinYahooURLCurrentDataSource.....
FinYahooURLCurrentDataSource.....72, 73, 115, 129, 131, 132, 138, 238, 252, 253, 289, 296, 297
FinYahooURLHistoricalDataSource.....
FinGoogleURLHistoricalDataSource.....117
FinYahooURLHistoricalDataSource...72, 73, 115, 116, 117, 120, 129, 130, 132, 138, 238, 252, 253, 289, 296, 297
FinYahooURLIntradayDataSource.....
FinYahooURLIntradayDataSource.....72, 73, 115, 116
Fixed Box Size.....
FIXED box size 153, 154, 223, 225, 226, 227, 231, 244, 245, 249
Fixed Box Size using ATR (Average True Range).....
Fixed Box Size using ATR (Average True Range) 226, 244
FloatingBarPlot.....
FloatingBarPlot.....94, 97, 113
Free Data Sources.....
free data sources.....5
Google Finance.....
Google Finance.....5, 6, 72, 73, 115

GraphObj
GraphObj....84, 85, 86, 102, 113
Grid
Grid....69, 81, 107, 108, 113
GroupBarPlot
GroupBarPlot....94, 98, 113
GroupDataset
GroupDataset....82, 83, 85, 113
GroupPlot
GroupPlot.93, 94, 95, 96, 97, 98, 99, 100, 101, 113
High-Low Method
High-Low method....232, 249
HistogramPlot
HistogramPlot....94, 99, 113
Image Rendering
BufferedImage....111, 113, 257, 263, 264, 265
Labels for Annotation
Labels for Annotation....4, 14, 60, 64, 195, 212
Legend
Legend....107, 110, 113
LegendItem
LegendItem....107, 113
Linear, Logarithmic, and Normalized y-axis scale
Linear, Logarithmic, and Normalized y-axis scale 3, 14
LinearAxis
LinearAxis....87, 92, 108, 113
LinearAxis....86
LinearScale
LinearScale....83, 113
LineGapPlot
LineGapPlot....94, 99, 113
Local File-Based Data Sources
Local File-Based Data Sources....121
LogAxis
LogAxis....86, 88, 92, 108, 113
LogScale
LogScale....83, 113
Markers
Marker. 59, 93, 102, 103, 105, 106, 107, 109, 110, 113, 114
Metastock
Metastock.5, 6, 7, 72, 73, 115, 116, 121, 122, 123, 124, 125, 127
Momentum
Momentum...4, 14, 46, 47, 49, 50, 56, 70, 71, 163, 167, 168, 169, 171, 191, 192, 274, 275
46
Mountain (or filled) line plot of close data
Mountain (or filled) line plot of close data....3
MouseListener
MouseListener....110, 111, 113
MoveData
MoveData....110, 113
MoveObj
MoveObj....110, 113
Moving Average Bands
Moving Average Bands…3, 13, 15, 38, 39, 40, 71, 145, 146, 158, 274, 275
Moving Average Convergence/Divergence (MACD)
Moving Average Convergence/Divergence (MACD)....4, 14, 56, 163, 179
MultiLinePlot
MultiLinePlot....94, 100, 113
NumericAxisLabels
NumericAxisLabels....91, 92, 113
NumericLabel
NumericLabel....108, 109, 113
OHLCPlot
OHLCPlot....15, 94, 100, 113
Open-High-Low-Close Bar plot
Open-High-Low-Close Bar plot....3
Open-High-Low-Close plots....15
OHLCPlot....15
Paid Data Sources
Paid data sources....5
Parabolic SAR
parabolic SAR....2, 3, 13, 15, 42, 44, 71, 158, 159, 164, 274, 275, 290, 297
Percentage Box Size
Percentage Box Size....224
PhysicalCoordinates
PhysicalCoordinates....83, 84, 85, 86, 113
Point And Figure Dialog Box
Point And Figure Dialog Box....227
Point and Figure plot
Point and Figure plot. 3, 15, 23, 78, 150, 153, 223, 229, 237, 251
Point And Figure Price Mode
Point And Figure Price Mode....232
Point and Figure y-Axis Scale Modes
Point and Figure y-Axis Scale Modes....223
Primary Chart Dialog
Primary Chart Dialog...4, 141, 148, 227, 234, 245, 250, 272, 276, 279
Printing
printing....iv, 111, 257, 258
QCChart2DNet.DLL
Newtonsoft....285, 293
- Quinn-Curtis core charting classes....285, 293
and Newtonsoft....285
QCSPCChartNet.DLL
QCTAChartNet.dll....7
QCTAChartNet.dll, QCChart2DNet.dll....285
QCChart2DNet.dll....285, 293
QCTAChartNet.dll – Quinn-Curtis financial technical analysis classes 285, 293
Quandl
Quandl 5, 6, 7, 68, 72, 73, 115, 116, 118, 119, 120, 121, 123, 125, 126, 127, 128, 131, 133, 135, 136, 137, 270, 277, 289, 297
Rate of Change (ROC)
Rate of Change (ROC)....4, 14, 49, 163, 171
Redistributable License
Redistributable License....ii
Regionalization
12. Regionalization....269
regionalization....iv, 269
267

Relative Strength (RSI)
Relative Strength (RSI)…4, 14, 51, 163, 173, 187, 274
Renko Chart Dialog Box
Renko Chart Dialog Box.....245
Renko plot
Renko Plot.....3, 15, 78, 150, 153, 237, 251, 272
Renko Price Mode
Renko Price mode.....249, 250
Renko y-Axis Scale Modes
Renko y-Axis Scale Modes.....244 244
Reversal Count
Reversal Count.....78, 150, 153, 154, 235, 273
Scatter Plots
scatter plots.....69, 81, 93, 101, 102, 105
Secondary Chart Dialog
Secondary chart dialog….163, 166, 168, 170, 179, 184
Simple line plot of close data
Simple line plot of close data.....3
Simple moving averages
simple moving averages 3, 13, 15, 34, 36, 143, 155
SimpleBarPlot
SimpleBarPlot.....105, 113
SimpleDataset
SimpleDataset.....82, 85, 113
SimpleLineMarkerPlot
SimpleLineMarkerPlot.....105, 106, 113
SimpleLinePlot
SimpleLinePlot.....105, 106, 113
SimplePlot
SimplePlot.....93, 105, 106, 109, 110, 113
SimpleScatterPlot
SimpleScatterPlot.....105, 106, 113
StackedBarPlot
StackedBarPlot.....94, 98, 113
StackedLinePlot
StackedLinePlot.....94, 101, 113
StandardLegend
StandardLegend.....107, 113
Stochastic (Fast and Slow)
Stochastic (Fast and Slow).....4, 14, 53, 163, 174
StringAxisLabels
StringAxisLabels.....91, 92, 113
StringLabel
StringLabel.....108, 109, 113
Synchronize scrolling (panning)
Synchronize scrolling (panning).....3, 14, 34
Templates
template.....2, 6
TickMark
TickMark.....111, 112, 113
TimeAxis
T imeAxis.....86
TimeAxis.....86, 89, 91, 92, 108, 113
TimeAxisLabels
TimeAxisLabels.....91, 92, 113
TimeCoordinates
TimeCoordinates.....83, 84, 113
TimeGroupDataset
TimeGroupDataset.....82, 83, 85, 113
TimeLabel
TimeLabel.....108, 109, 113
TimeScale
TimeScale.....83, 113
TimeSimpleDataset
TimeSimpleDataset.....82, 85, 113
ToolTips
DataToolTip.....110, 113
Traditional Box Size
traditional box size.....223
Trend lines
trend lines.....4, 9, 14, 60, 195
Trial License
Trial License.....ii
True Range
True Range.....42, 45, 147, 226, 227, 244, 245
Tutorial
tutorial.....iv, 7, 9, 116
Typical Price Method
Typical Price Method.....233, 250
URL-Based Data Sources
URL-based data sources.....117, 121
UserControl
UserControl.....69, 81, 82, 113, 283
UserTA.....287
UserCoordinates
UserCoordinates.....83, 113
Visual Basic.....9
Visual Basic.....7, 9, 260, 291 291
Visual C#.....9
Visual Basic.....291
Visual C#.....9, 283, 291
Volume charts
Volume charts.....4, 14, 58, 163, 182 182
Williams %R
Williams %R4, 14, 55, 56, 163, 177, 179, 274, 275
Windows Applications
Windows Applications.....9, 283
WorkingCoordinates
WorkingCoordinates.....83, 84, 113
WorldCoordinates
WorldCoordinates.....83, 84, 113
Yahoo Finance
Yahoo Finance.....5, 6, 73
zooming
Zoom.....110, 113, 114
zooming.....3, 14, 26, 34, 70, 110, 141, 237, 251 114
.....
.....
.....
.....
49, 72, 74, 75, 82, 83, 86, 91, 94, 99, 101, 111, 113, 114, 115, 116, 119, 122, 123, 136, 149, 152, 153, 156, 157, 185, 193, 198, 202, 206,

212, 214, 217, 219, 220, 238, 258, 259, 264
Data Sources..... " 15
data sources. iv, 5, 7, 72, 115, 116, 117, 119, 120, 121, 123, 131, 133, 289, 297